AUTOMATED ALERTING FOR BLACK HOLE ROUTING

by

Vinay Puri

September 2007

Thesis Co-Advisors:	Geoffrey Xie J. D. Fulp

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Automated Alerting for Black Hole Routing			**5. FUNDING NUMBERS**
6. AUTHOR(S) Vinay Puri			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Distributed/Denial of Service (D/DoS) attacks are the most common and easy-to-launch attacks against a computer or network. Once a D/DoS attack is recognized, there are several methods available to mitigate its impact. One of the methods is to drop the attacker's traffic at the edge of the network via Null Routing-also called Black Hole Routing (BHR). BHR is more efficient than the creation and processing of access control lists. Prior work has validated the effectiveness of BHR in mitigating D/DoS attacks in a setting where the defense is activated manually. This research built upon that work and developed a proof-of-concept automated BHR process integrated with Snort, an open source Intrusion Detection System (IDS), to facilitate a faster reaction to a D/DoS attack. A real test bed consisting of Cisco routers was created to evaluate the performance of the developed system. The results demonstrated that the automation of BHR is both possible and desirable in mitigating D/DoS attacks.

14. SUBJECT TERMS: Automated Alerting for Black Hole Routing, Black Hole Routing, Intrusion Detection System(IDS), Automation of Black Hole Routing, DDoS Attacks, Network Security, ISP Network Security, Snort , Null Routing, Customer Triggered Black Hole Routing, Mitigating DDoS attacks, BGP, iBGP.			**15. NUMBER OF PAGES** 139
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

AUTOMATED ALERTING FOR BLACK HOLE ROUTING

Vinay Puri
Squadron Leader, Indian Air Force
B.E., Manipal Institute of Technology, Manipal, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION WARFARE

and

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Vinay Puri

Approved by: Geoffrey Xie, Ph.D.
Thesis Co- Advisor

J. D. Fulp
Thesis Co-Advisor

Dan C. Boger, Ph.D.
Chairman, Department of Information Sciences

Peter J. Denning, Ph.D.
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Distributed/Denial of Service (D/DoS) attacks are the most common and easy-to-launch attacks against a computer or network. Once a D/DoS attack is recognized, there are several methods available to mitigate its impact. One of the methods is to drop the attacker's traffic at the edge of the network via Null Routing-also called Black Hole Routing (BHR). BHR is more efficient than the creation and processing of access control lists. Prior work has validated the effectiveness of BHR in mitigating D/DoS attacks in a setting where the defense is activated manually. This research built upon that work and developed a proof-of-concept automated BHR process integrated with Snort, an open source Intrusion Detection System (IDS), to facilitate a faster reaction to a D/DoS attack. A real test bed consisting of Cisco routers was created to evaluate the performance of the developed system. The results demonstrated that the automation of BHR is both possible and desirable in mitigating D/DoS attacks.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I thank my advisor, Professor Geoffrey Xie, for providing me such an interesting research topic. He showed me different ways to attack the problem and was very generous in understanding the requirements throughout the research. He even ensured that I received the proper knowledge about every component of the research topic to provide better output. When required, he even went out of his way to get me a travel grant.

Eight months is a long time, from laying the initial plans to finally achieving the goal. In this entire journey, one person who was very instrumental in channelizing my thoughts was my co-advisor, J. D. Fulp. He never let me deviate from my focal point. Special thanks go to him for his solid contributions during my entire journey. He even taught me how to bring my thoughts to paper and helped me in setting an initial platform.

Besides being indebted to both my advisors, I would like to thank Professor John Gibson for providing me the necessary equipment to set up the real-time test bed. Also, I would like to thank Breen Dix for his help in editing the draft of my thesis.

My special thanks also go to the Indian Air Force, who gave me the opportunity to study in this prestigious institute. Last but not least, I would like to thank my father, who, sitting in heaven, has been my strength throughout. I also acknowledge my mother, my brother, and my uncle for giving me unconditional, absolute, and 101% support toward achieving the main goal and not allowing me to worry about things at home. They encouraged me throughout to achieve the purpose that I was here for. Without their unconditional support, I would not have achieved this task.

I. INTRODUCTION

The research in this thesis extends the research done by Nikolaos Stamatelatos, in which he stress-tested three implementations of the black hole routing (BHR) concept in a lab environment.[1] In the "Future Work" section of Mr. Stamatelatos' research, he indicated:

> As noted previously in this study, we assumed that a given DDoS attack had been positively identified by either an automated system or a human operator. The ability to automatically identify an attack using an IDS/IPS system would greatly improve the performance of BGP Black Hole Routing. As the research in this field to date is limited, it is our first suggested area for future work.

Thus, the work of this thesis set about not only selecting and configuring an appropriate IDS to detect a distributed denial of service (DDoS) attack; but to also integrate this detection capability into an enhanced BHR system, by having the IDS directly cue the "trigger router" that sends the null—black hole—route update to all border routers. The result is a working implementation of a fully automated *attack-detect-react-protect* BHR system.

Chapter I introduces the underlying concepts of denial of service (DoS), and DDoS attacks. Since the BHR protective mechanism can be used to defend against either of these, the collective abbreviation D/DoS will be used henceforth to indicate either the single-source (DoS) or multiple-source (DDoS) form of attack.

Chapter II reviews the three methods of null route association (i.e., *by source*, *by destination*, and *by customer*) as described by Stamatelos.[2] This review not only helps the reader to understand BHR, but is also pivotal to understanding why the *by destination* method was chosen for the automated "alerter" enhancement.

Chapter III describes the Systems Engineering approach that was taken while researching and developing the automated BHR implementation.

[1] Nikolaos Stamatelatos, "A Measurement Study of BGP Black Hole Routing Performance," Master's Thesis, September 2006.

[2] Ibid.

Chapter IV details how the IDS was chosen, configured, and evaluated.

Finally, Chapter V provides a summary and suggests additional BHR-related research.

A. DISTRIBUTED/DENIAL OF SERVICE ATTACKS

DoS attacks include any intentional, malicious means by which the legitimate consumers of information are denied (or adversely delayed) access to needed information. Typical methods of DoS include either the outright destruction (i.e., “hard kill”) of the information or systems that store, transmit, or process the information; or the overwhelming (i.e., “soft kill”) of system resources that are employed to store, transmit, or process the information. DoS attacks, as distinguished from DDoS attacks, are executed from a single source. DDoS attacks amplify the effect of a DoS attack by having the underlying malicious actions executed/launched from many sources.

Figure 1 provides example illustrations of both a DoS and a DDoS attack. Also illustrated is a slightly modified DDoS that employs “reflectors.” Unlike “slave” systems that are non-complicit in the attack but have nonetheless been infected/compromised via the installation/introduction of remote-controlled code, “reflector” systems are being used “normally.” For example, a “slave” may have Trojan software installed that “listens” on port 34,565 for specially formatted commands from the “master” system and then acts on those commands. A “reflector,” on the other hand, will be any system that is behaving as per normal protocols; however, the manner in which the protocols are being used serves to adversely affect the target system. For example, the “master” may send a message to its “slaves” to ping attack 192.168.10.35 at 1015EDT on 26 September. The “slaves” will wait until the prescribed date and time, then spoof the target machine’s Internet protocol (IP) address and send hundreds of Internet Control Message Protocol (ICMP) echo requests (pings) to a list of known/suspected “reflector” systems. The “reflectors,” in turn, echo-reply (normal behavior) to the legitimate owner of the IP address (the target), and thus degrade/overwhelm not only the target, but possibly the network(s) the target host is attached to as well.

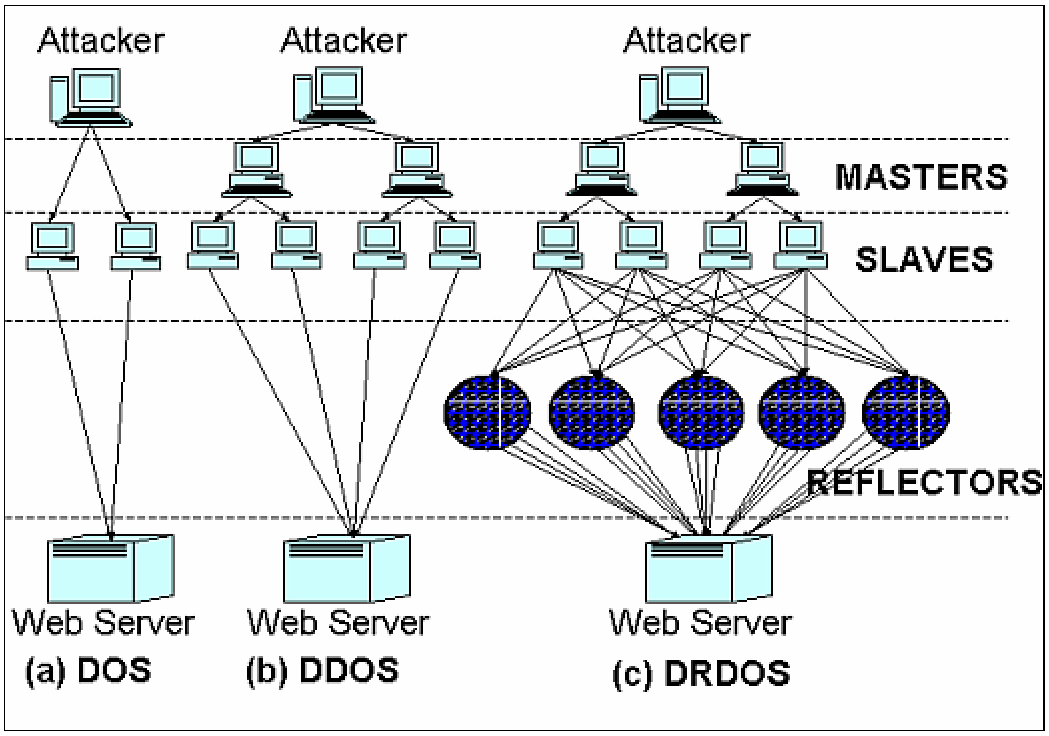


Figure 1. (A) DoS attack, (B) DDOS attack, (C) DRDOS (Distributed Reflection Denial Of Service) or DDOS with Reflectors.[3]

B. BLACK HOLE ROUTING

Black hole routing is a clever way of saying "route this packet to the trash." The concept is quite simple and leverages the basic operation of routers. Routers inspect received packets to identify the destination network (or host, if the destination network is directly attached), then consult their routing table to see if there is either a specific next hop for the that network or, as an alternative, a default next hop to reach that network. This route table also called a router's Forward Information Base (FIB), is populated both by manual static route configuration and by the automated sharing of information via dynamic routing protocols.

A black hole route tells the router to send the subject packet to the *null0* interface (a non-existent interface), which is equivalent to telling the router to "route this packet to the trash." As will be discussed in Chapter II, this "route to the trash" idea can cue off of

[3] Yanet Manzano, "Tracing the Development of Denial of Service Attacks: A Corporate Analogy," ACM Student Magazine, www.acm.org/crossroads/xrds10-1/tracingDOS.html, last accessed 7 July 2007.

a specific sender (source IP address), or a specific destination (destination IP address). Figure 2. offers a simplistic model that suggests the immediate network relief that would be realized if any/all of several key routers (i.e., those directly facing zombies) were told to “black hole” any packets going *to* the victim IP address (i.e., destination-based BHR). In general practice, the “key” routers will be those positioned at the border of the victim’s autonomous system (AS).

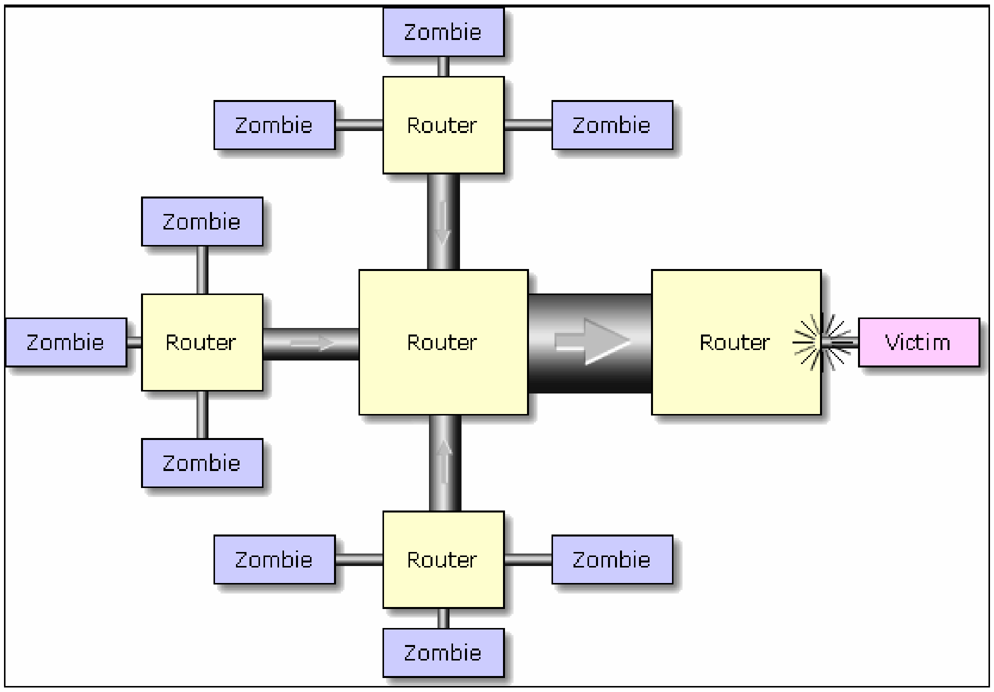


Figure 2. Distributed zombie traffic aggregation.[4]

C. ADVANTAGES OF BHR

Undoubtedly, automating BHR has substantial advantages. Prior to the discussion of automation and its advantages, it is essential to know the following characteristics of BHR, which makes our argument to automate the BHR process stronger.

In networking, the Access Control List (ACL) is the list of rules which decides the treatment to be given to incoming/outgoing packets. An ACL is a method by which

[4] Steve Gibson, “Distributed Reflection Denial of Service,” Gibson Research Corporation, www.grc.com/dos/drdos.htm, last accessed 7 July 2007.

packets can be dropped/forwarded for the inbound/outbound traffic depending upon the rules. An ACL is required to be configured on the device where the filtering of packets is desired. In network security fields, routers, servers, individual hosts, etc., can have an ACL. Although ACLs are effective, they are costlier than BHR in terms of processing. During a D/DoS attack, when thousands of packets are flooded towards the victim machine, processing an ACL can consume central processing unit (CPU) resources. Unlike an ACL, the processing power required to implement null routing is minimal because typically routing table lookup is assisted by special hardware.[5]

Figure 3 illustrates the advantages of null routing vis-à-vis ACL. This figure also portrays that packets destined for the null route are directly thrown to the “bit-bucket.”; i.e., they are discarded.

A second advantage is that a single entry can be used to control access to both inbound and outbound packets.

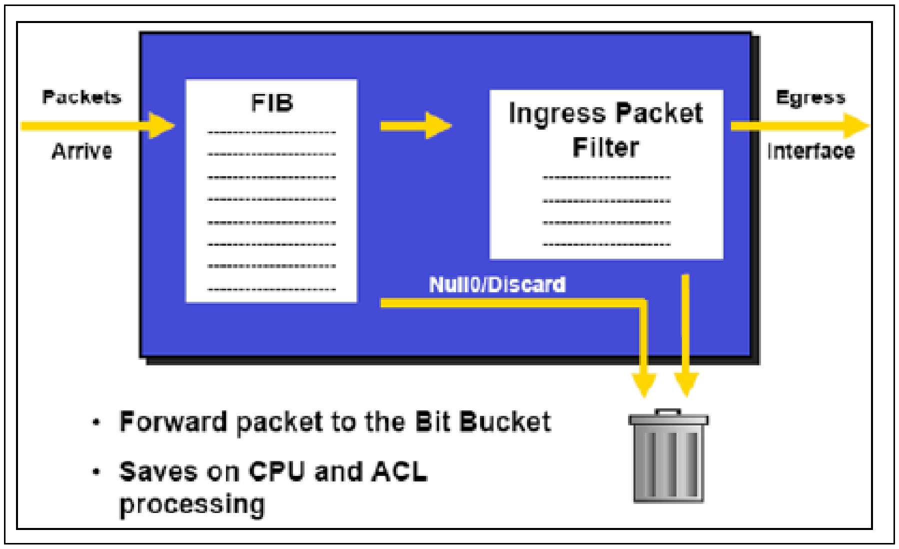

Packets
Arrive
FIB
Ingress Packet Filter
Egress
Interface
Null0/Discard
• Forward packet to the Bit Bucket
• Saves on CPU and ACL processing

Figure 3. Advantage of the BHR/Null0 interface.[6]

[5] Nanog, The North American Network Operators' Group, www.nanog.org/mtg-0110/ppt/greene.ppt, last accessed 8 July 2007.

[6] Ibid.

The third advantage is that a null route can be redistributed via routing protocols such as BGP and, as such often only a single black hole route needs to be entered on a router. This advantage is elaborated in detail in Appendix A.

D. PREVIOUS RESEARCH

There have been a couple of studies carried out that talk about the analysis of BHR. The first is M. Kleffman's thesis, "Analysis of Effects of BGP Black Hole Routing on a Network like the NIPRNET," and a second study carried out by N. Stamatelatos in his thesis, "A Measurement Study of BGP Blackhole Routing Performance."

Kleffman used simulation as the evaluation technique for his research. A simulation software package, Opnet Modeler version 10.5, was used by Kleffman to perform analysis. The performance metrics chosen by Kleffman was queuing delay, latency, router convergence delay, and bandwidth utilization.

Stamatelatos, on the other hand, used a real test-bed network to evaluate the effectiveness of various methods of BHR. The performance metric chosen by Stamatelatos was router response time, router CPU load, and link load. The following sub-sections briefly discuss the network setup followed for both previous researches.

1. Lab Setup/Test Bed

a. Lab Setup in Kleffman's Research

As stated earlier in this section, the evaluation method followed by Kleffman was simulation. Kleffman simulated 39 experiments, which were replicated five times. His simulated network consisted of six border routers, a trigger router, and twelve customer routers. He collected and analyzed the data under different workloads (i.e., the amount of traffic). Kleffman's main goal was to determine the following by simulating experiments:

- To check the effectiveness of BHR in defending the nonsecure Internet Protocol Router Netowrk (NIPRNet) against DDOS attacks.
- To check the effectiveness of BHR without all the border routers participating.
- To study the comparison of remote-triggered versus customer-triggered BHR.

b. Lab Setup in Sramatelos' Research

Stamatelatos evaluated the performance of BHR methods in the lab with three real-time test-bed networks. He selected seven routers to simulate the various environments that depict the real-time AS. Stamatelatos utilized his chosen performance metrics in his test beds. A brief discussion of the three test-bed networks used by him is as follows:

(1) Test-Bed Network #1. The main task of this test-bed network was to evaluate the performance of both the methods of remote-triggered BHR, i.e., destination-based and source-based. Remote-triggered BHR is reviewed in Chapter II. Stamatelatos simulated an AS environment with three border routers, two internal routers, and one trigger router. In this test bed, malicious traffic would approach the AS from different sources. In addition, traffic would also traverse through different border routers. Once the attack began, the trigger router inside the AS was configured to advertise either source-based or destination-based BHR to evaluate both the techniques.

(2) Test-Bed Network #2. The main purpose of test-bed network #2 was to evaluate customer-triggered BHR and then compare its performance with remote-triggered BHR. He simulated this test bed by maintaining the same topology as discussed in test-bed network #2. The only difference was the positioning of the trigger router. The trigger router was placed in line with the target-host to simulate the customer network.

(3) Test-Bed Network #3. The purpose of test-bed network #3 was to evaluate the performance of BGP BHR in a network where the routers have sufficient CPU capacity but some of the internal links of the victim's network become congested during an attack. He simulated this to evaluate performance when the limiting

factor could be the link load, not the router CPU load. He utilized five routers, one of which is Juniper router with a relatively high CPU capacity, to simulate this test bed. The topology was different from test-bed networks #1 and #2.

2. Research Conclusions

a. Kleffman's Research Conclusions

- No adverse effects due to BHR have been noticed on the normal operations of network like NIPRNet.
- BHR proved to be successful in defending network like NIPRNet under D/DoS.
- BHR is more effective when all the border routers of AS are participating in BHR.
- Remotely-triggered BHR is more effective as compared to customer-triggered BHR.

b. Stamatelatos' Research Conclusions

- Resource overload may disrupt the BGP session between the trigger router and a border router and thus degrade the performance of BGP BHR.
- He seconded the opinion of Kleffman's conclusion that customer-triggered BHR is not as effective as other techniques.
- Destination-based BHR performed best in his test-bed simulations.
- BHR would be totally inefficient if applied 40 seconds or more after the DDoS attack initialization (especially with high link load).

E. COMMENTS

Kleffman's main focus was NIPRNet and his research results were based on simulation. The major disadvantage of simulation is that these applications are highly dependent on the computer configurations where they are installed. Stamatelatos overcame both these flaws by utilizing real-time test-beds in his research. The common shortcoming in both studies was the absence of IDS/IPS. Both Kleffman and Stamatelatos assumed in their research that a D/DoS attack has previously been recognized. Another shortcoming was the absence of automation. In both studies, they manually added the static route to the trigger router to advertise the null route. These shortcomings have been overcome in this research by engaging the services of IDS and automating the process of null route advertisement.

Another major shortcoming in both previous researches was in their conclusions. Kleffman and Stamatelatos both concluded that customer-triggered BHR is least effective.

Kleffman stated[7]:

> Customer-triggered black hole routing is clearly not as effective as remote-triggered black hole routing. This is due to BGP updates being sent via TCP packets and the communication links between the bases and border routers not being of sufficient size to handle the same amount of traffic as the communication links between the border routers.

On the other hand, Stamatelatos stated:

> Second, we concluded that, of the three basic BGP Blackhole routing methods, the customer-triggered method has the worst performance.

Neither explored how this technique could be effective. Although the main goal of our research was not to investigate which BHR method was best or most effective, in Chapter IV, we have nevertheless discussed two methods by which the customer-triggered method can be as effective as other techniques. Besides, by employing the method we suggest in Chapter IV, Stamatelatos' fourth conclusion, as described above, is nullified.

[7] Michael D. Kleffman, "Analysis of Effects of BGP Black Hole Routing on a Network like the NIPRNET." Master's Thesis, AFIT, 2005.

THIS PAGE INTENTIONALLY LEFT BLANK

II. METHODS FOR BHR

This chapter discusses three things. First, it discusses how the network is prepared for BHR. Second, it briefly touches upon the various methods available to achieve BHR. The final section discusses the motive for automating one of the methods.

A. INITIAL PREPARATION FOR ACHIEVING BHR

It is essential that Internet Service Providers (ISPs) or organizations (in the case of customer-based BHR) have some initial preparation ready before one of the BHR methods is implemented to mitigate a D/DoS attack. There are four steps that need to be considered for this initial preparation. It is worth mentioning that these steps have no impact on the operation of the network.

The first step is to set up a static route to the Null0 interface on all of the routers that will be triggered to implement BHR. Included in this step is the allocation of a block of IP address space that is not used on the Internet [RFC 1918]. For example, the IP address 192.168.10.0/24 was used by us in our lab. The static route added in all the routers appears as follows:

```
ip route 192.168.10.0 255.255.255.0 Null0
```

This static route indicates that any packet destined for 192.18.10.0/24 will be black-holed into the bit bucket.

The second step is to set up one router or host as the trigger router or the 'Black Hole Route Server' in the network. This router will be pushing out BGP blackhole route announcements (both addition and removal) for the victim hosts.The announcements are triggered when specific static routes are added or removed on the trigger router. The trigger router does not have to be a dedicated router. On a Cisco router, this equates to redistribution edits, a route map, and some static routes with tags.[8] A detailed configuration of a trigger router and its explanation is attached as Appendix A.

[8] Cisco, "Remotely Triggered Black Hole Filtering—Destination Based and Source Based," Cisco Press, www.cisco.com/warp/public/732/Tech/security/docs/blackhole.pdf, last accessed 9 July 2007.

The third step activates the BHR. In this step, specially tagged static routes with customized destination address (e.g. that of the victim) and specific next hop such as "192.168.10.1" are configured on the trigger router. These routes are redistributed into the BGP protocol and subsequently pushed into the FIBs of the BGP-speaking border routers. Finally, the border router start to black-hole the attack packets because of the combined effort of the new static routes and the preconfigured null routes. This step is also covered in Appendix A.

As the final step, it is important to remove the above static routes from the trigger router when the D/DoS attack is over.

B. METHODS FOR BHR

There are two basic methods via which BHR can be achieved. These methods are remote-triggered BHR and customer-triggered BHR. Remote-triggered BHR is the basic version of BHR and can further be categorized as destination-based or source-based routing depending upon the IP address information used to block traffic. The following three sub-sections briefly discuss these three methods of BHR.

1. Remote-Triggered Destination-Based BHR

In remote-triggered destination-based BHR, the destination IP address which is under attack is advertised to be black-holed. The four steps discussed in the previous section of this chapter are prerequisites to achieving this technique. The following excerpt from a Cisco white paper nicely explains remote-triggered destination-based BHR.[9]

> The challenge is to find a way to quickly drop the offending traffic at the network edge, document and track the black-holed destination addresses, and promptly return these addresses to service once the threat disappears. Destination-based IP black hole filtering with remote triggering allows a network-wide destination-based black hole to be propagated by adding a simple static route to the triggering device (trigger). The trigger sends a routing update for the static route using iBGP to the other edge routers

[9] Cisco, "Remotely Triggered Black Hole Filtering—Destination Based and Source Based," Cisco Press, www.cisco.com/warp/public/732/Tech/security/docs/blackhole.pdf, last accessed 9 July 2007.

configured for black hole filtering. This routing update sets the next hop IP address to another preconfigured static route pointing to the null interface.

This process is illustrated in Figure 4.

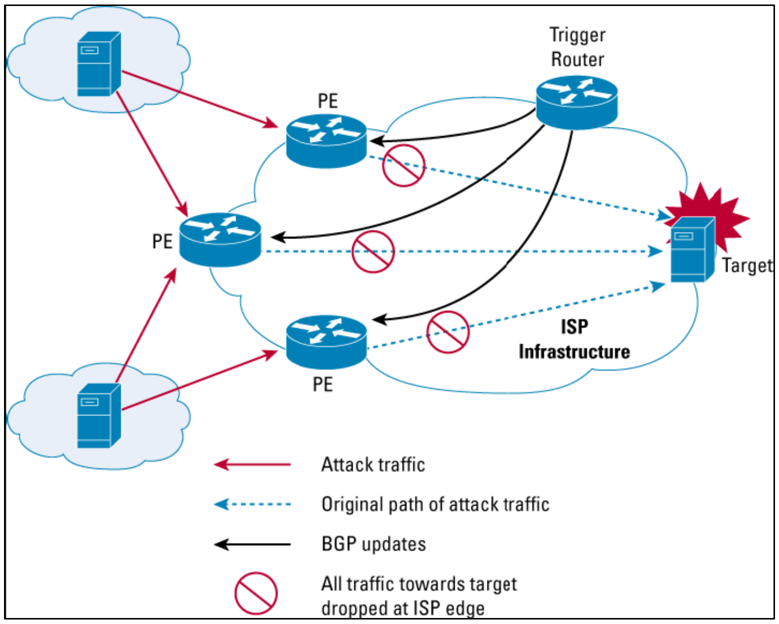

Trigger Router
PE
PE
PE
Target
ISP Infrastructure
Attack traffic
Original path of attack traffic
BGP updates
All traffic towards target dropped at ISP edge

Figure 4. Remote-triggered destination-based BHR.[10]

The key benefit of this technique is that packets are dropped at the edge of the network itself and thus other traffic within the AS is not affected.

2. Remote-Triggered Source-Based BHR

As the name suggests, the source IP address(s) from where the attack is originated are advertised by trigger router to be black-holed. The four-step initial preparation, as discussed earlier in this chapter, is mandatory. Implementation of this method is very similar to destination-based but it has its own pros and cons. The following excerpt from the Cisco documentation explains source-based BHR and discusses the advantages of this technique.

[10] Cisco, "Remotely Triggered Black Hole Filtering—Destination Based and Source Based," Cisco Press, www.cisco.com/warp/public/732/Tech/security/docs/blackhole.pdf, last accessed 9 July 2007.

Source-based BHR provides the ability to drop traffic at the network edge based on a specific source address or range of source addresses. The idea here is to drop the traffic at the edge based on the source address if the source of the attack can be identified. The challenge here is to find out whether the source is spoofed or not but on the other side the advantage is unlike Destination-Based BHR, entire traffic destined for target host will not be dropped. This kind of BHR technique would permit legitimate traffic from other sources to reach the target. Implementation of source-based black hole filtering depends on Unicast Reverse Path Forwarding (URPF), most often loose mode URPF.Loose URPF checks the packet and forwards it if there is a route entry for the source IP of the incoming packet in the router FIB. If the router does not have an FIB entry for the source IP address, or if the entry points to Null0, the Reverse Path Forwarding (RPF) check fails, and the packet is dropped.

Figure 5 illustrates this process of remote triggered source based BHR.

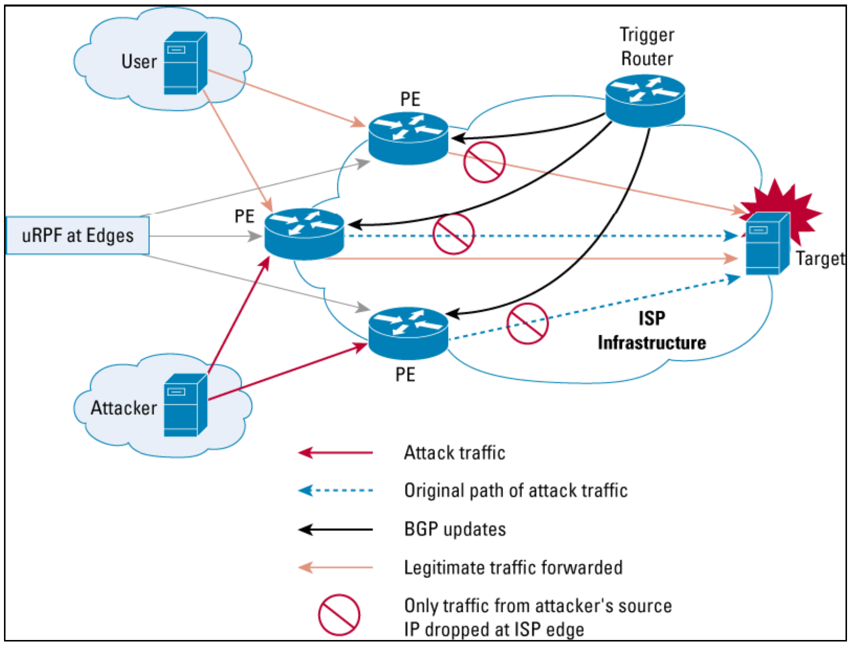

User
Trigger Router
PE
PE
uRPF at Edges
Target
ISP Infrastructure
PE
Attacker
Attack traffic
Original path of attack traffic
BGP updates
Legitimate traffic forwarded
Only traffic from attacker's source IP dropped at ISP edge

Figure 5. Remote-Triggered Source-Based BHR.[11]

The key benefit in this technique is the same as it is with destination-based BHR—the packets are dropped at the edge of the network. Besides, if the source is identified to be non-spoofed, then this technique is the most effective.

[11] Cisco, "Remotely Triggered Black Hole Filtering—Destination Based and Source Based," Cisco Press, www.cisco.com/warp/public/732/Tech/security/docs/blackhole.pdf, last accessed 9 July 2007.

3. Customer-Triggered BHR

Unlike remote-triggered BHR, where the control is with the ISPs, customer-triggered BHR is controlled by the customer of the ISP. In our scenario, the customer would be the military network under attack. Customer-triggered BHR takes advantage of the fact that the customer's router speaks BGP to the border routers. If the network administrator notices that his/her network is being overloaded by a D/DoS attack, he/she can send an iBGP update to the border routers through his/her local router to have the incoming traffic dropped. As with remote-triggered black hole routing, the border routers would have to be configured to accept and apply the BGP route updates from the customer's router.

Customer-triggered BHR is not widely popular on the Internet because commercial ISPs do not like to give a customer the ability to write FIB entries at the ISP's border routers. If a customer is compromised, there is a prominent threat to the border routers of the ISP. Still, this kind of BHR technique can bring enormous advantages, especially in the military networks, because it is easier to establish the trust relationship and pre-agreement between ISPs and military networks than with other settings. In this thesis, I have simulated an automation of customer-based BHR in which once the destination under attack is recognized, it is black-holed until the D/DoS attack is mitigated. Another advantage of this kind of BHR technique is that once we are sure that the attack is from a specific source, we can black-hole the source also. All that is required is to make the changes in the detector (IDS) and automate the BHR process accordingly. The IDS chosen in our scenario is covered in greater detail in Chapter IV. The unique network setup chosen to achieve customer-based BHR is shown in Figure 6.

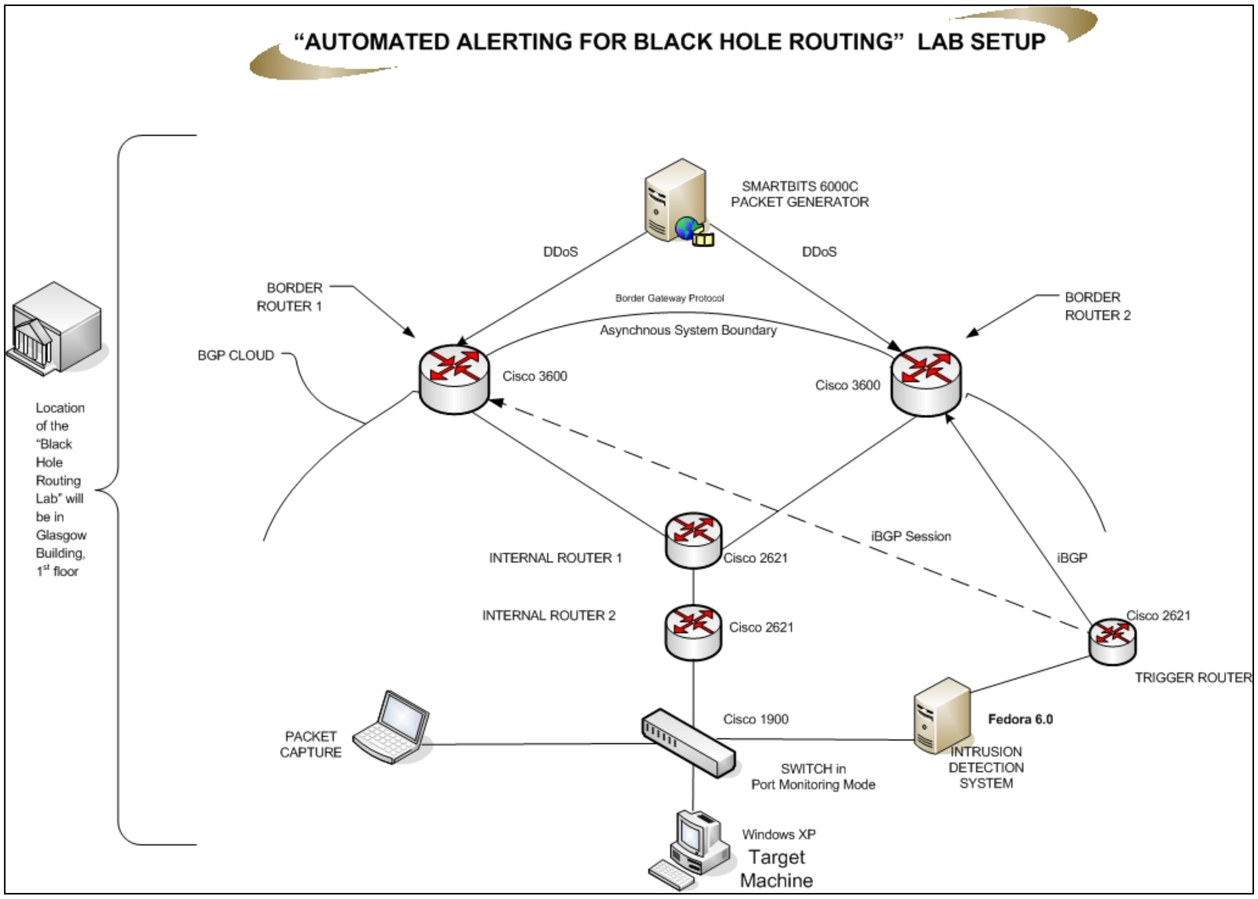

"AUTOMATED ALERTING FOR BLACK HOLE ROUTING" LAB SETUP
SMARTBITS 6000C PACKET GENERATOR
DDoS
DDoS
BORDER ROUTER 1
BORDER ROUTER 2
Border Gateway Protocol
Asynchnous System Boundary
BGP CLOUD
Cisco 3600
Cisco 3600
Location of the "Black Hole Routing Lab" will be in Glasgow Building, 1st floor
INTERNAL ROUTER 1
Cisco 2621
iBGP Session
iBGP
INTERNAL ROUTER 2
Cisco 2621
Cisco 2621
TRIGGER ROUTER
Cisco 1900
Fedora 6.0
INTRUSION DETECTION SYSTEM
PACKET CAPTURE
SWITCH in Port Monitoring Mode
Windows XP Target Machine

Figure 6. Lab setup simulating customer-triggered BHR.

C. AUTOMATING BHR AND ITS ADVANTAGES

Some of the proposed solutions have simulated the environment of D/DoS attacks to clients, in which, after the attack is detected, the network administrator manually configures a trigger router which then activates the border routers, via an iBGP update message, to perform null routing. In a real world scenario it is very difficult to reliably safeguard computer networks using this manual methodology, owing to the slow response cycle.

After the automation of BHR, there should be significant improvement in the response time. At present, after identification of the attack, the network administrator has to manually add one or more static routes to the trigger router. This significantly delays the response time. Other disadvantages with the manual method include the network administrator forgetting to configure the router, missing the receipt of an attack alert, or taking too much time to gain access to the trigger router. These issues can be drastically improved by automating the BHR process.

Automation of BHR would facilitate much faster reaction to D/DoS attacks, thus further embellishing the protective effectiveness of the null routing idea that is at the center of the success of the BHR strategy.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SYSTEM ENGINEERING APPROACH: AUTOMATING BHR (METHODOLOGY FOLLOWED)

> **System Engineering [Approach]**: An Interdisciplinary collaborative approach to derive, evolve, and verify a life-balanced system solution which satisfies customer expectations and meets public acceptability.[12]

It is stated by the International Council on System Engineering (INCOSE) that "System Engineering considers both the business and the technical needs of all customers, with the goal of providing a quality product that meets the user needs". Any engineering system can have various approaches to achieve the end result. Similarly, for achieving this thesis' primary objective of implementing an automated alerting process, this requirement can be addressed either by a top-down approach or bottom-up approach. With the system engineering approach, various complexities can be managed.

A. ADDRESSABLE REQUIREMENT USING THE TOP-DOWN APPROACH

The formal definition of the top-down approach in system engineering, as defined in Wikipedia, is as follows:

> In a top-down approach an overview of the system is first formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements. A top-down model is often specified with the assistance of "black boxes" that make it easier to manipulate. However, black boxes may fail to elucidate elementary mechanisms or be detailed enough to realistically validate the model.[13]

[12] Institution of Electrical and Electronics Engineers, "IEEE P1220, Standard for Application and Management of the System Engineering Process," New York, NY, 1994, p. 11.

[13] Wikipedia, http://en.wikipedia.org/wiki/Top-down, last accessed 9 July 2007.

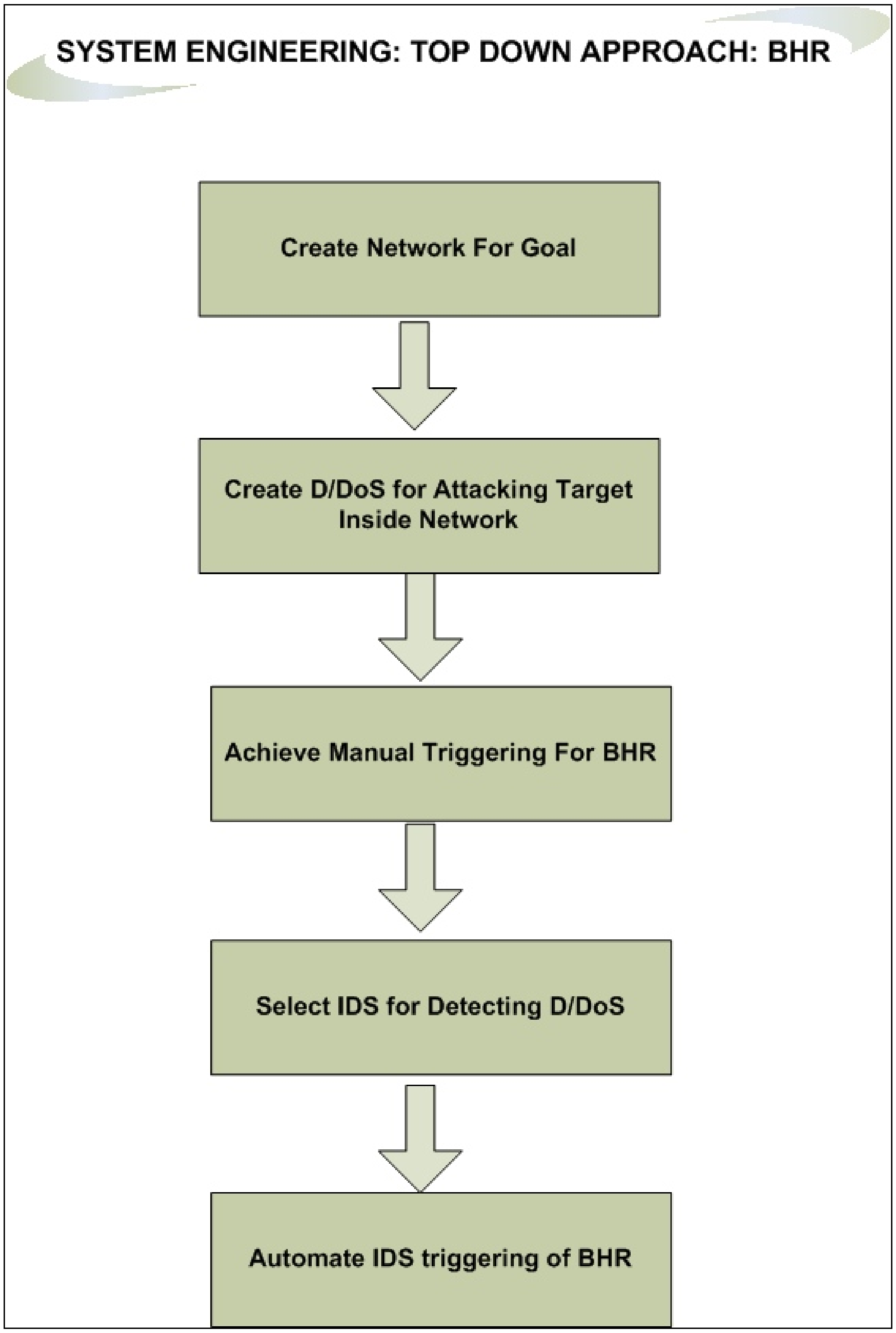

SYSTEM ENGINEERING: TOP DOWN APPROACH: BHR
Create Network For Goal
Create D/DoS for Attacking Target Inside Network
Achieve Manual Triggering For BHR
Select IDS for Detecting D/DoS
Automate IDS triggering of BHR

Figure 7. Top-down approach for achieving automated alerting for BHR.

The top-down methodology which can be adopted in this research is enumerated below and illustrated in Figure 7:

- Set up network environment to achieve BHR.
- Set up D/DoS attack scenario to suit our goal.
- Test the manual triggering of BHR by manually entering the static route.
- Detect D/DoS using the IDS.
- Achieve IDS triggering of BHR.

B. ADDRESSABLE REQUIREMENT USING BOTTOM-UP APPROACH

Similarly, in a bottom-up approach, every element of the system is stated explicitly in detail (e.g., which routers will be used to implement the BHR process). These elements are then linked together to form larger subsystems, which are then combined with other subsystems to form the final top-level system (i.e., the ultimate goal/ objective of the project). This strategy often resembles a “seed” model, whereby the beginnings are small, but eventually grow in complexity and completeness.[14] In this project, I have used the bottom-up approach.

The next few paragraphs will expound upon the notion of the bottom-up approach. The methodology employed when using this approach is as follows and is further illustrated in Figure 8.

First, the hardware and software needed to simulate the identical network scenario had to be collected. Once this minimum requirement was met, network connection tests were carried out to ensure its correct operation. The components used and the IP address scheme followed for the tests are discussed in detail in Chapter IV.

Second, hardware and software tools were identified to create the D/DoS attack scenario. Once the tools were in place, the attack was tested by sending packets from the source to the destination machine.

[14] Wikipedia, http://en.wikipedia.org/wiki/Bottom-up, last accessed 9 July 2007.

With the network and attack scenario created, the next task was to check the manual triggering of the BHR and measure how quickly the trigger router could advertise the null route.

Next, thorough research was carried out to find an appropriate alerting system to cue the trigger router alert in case of an attack. Research was conducted to find an open source/trial IDS that could be customized as necessary to implement the automated alerting.

Once the alerter had been selected and configured, the next requirement was to bring the alerter into the network setup and configure the IDS to generate D/DoS alerts, and to ensure that the alerts were actually generated.

Finally, the alerts generated by the alerter (IDS) were used to trigger the trigger router in order to complete the automation process.

In the bottom-up approach, further complexities could be accommodated into this environment to suit local/organizational requirements. For example, critical alerts could be sent via a message/e-mail to a network administrator, or other appropriate recipients, in addition to the triggering router.

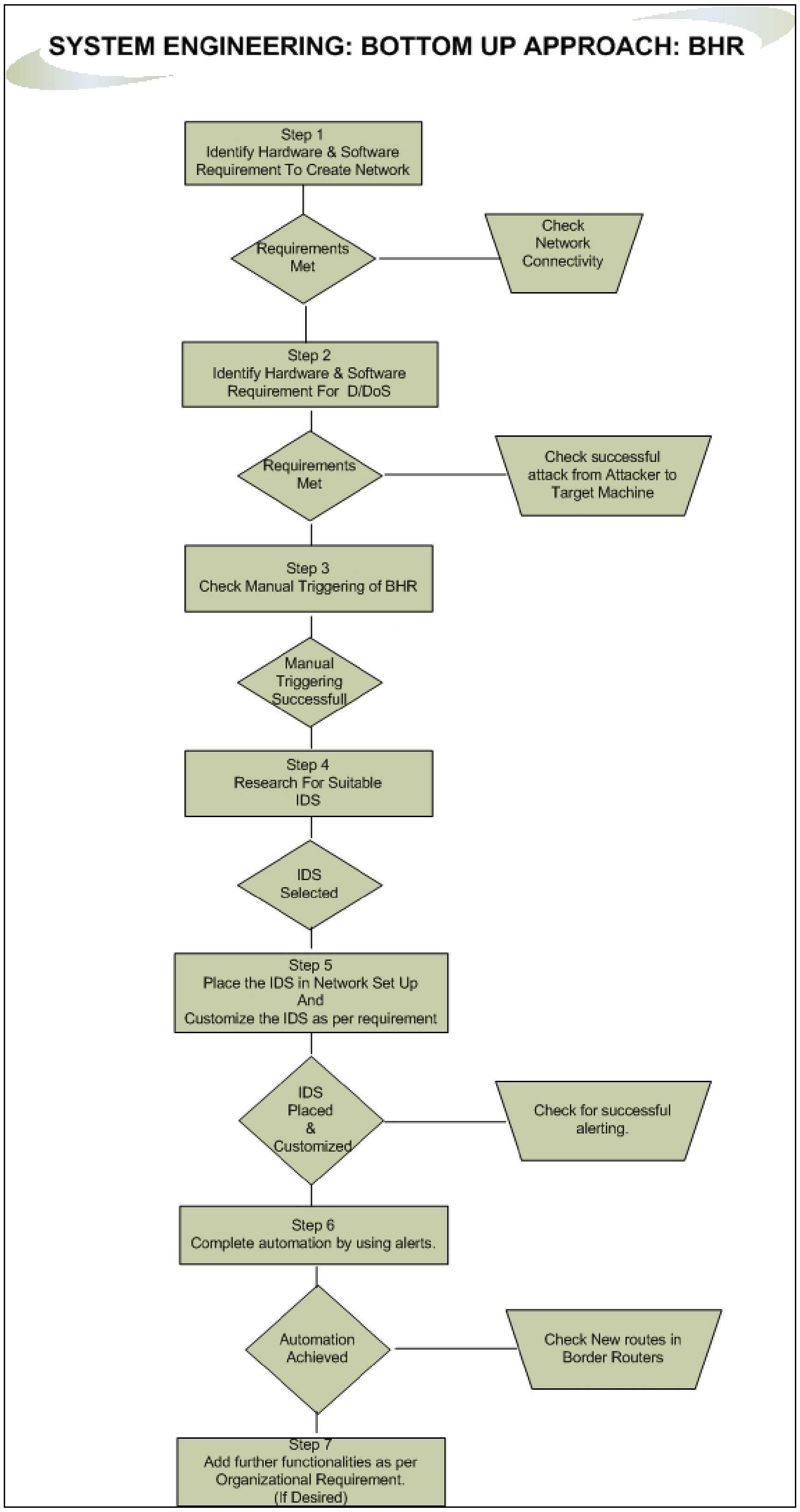

SYSTEM ENGINEERING: BOTTOM UP APPROACH: BHR
Step 1
Identify Hardware & Software Requirement To Create Network
Requirements Met
Check Network Connectivity
Step 2
Identify Hardware & Software Requirement For D/DoS
Requirements Met
Check successful attack from Attacker to Target Machine
Step 3
Check Manual Triggering of BHR
Manual Triggering Successfull
Step 4
Research For Suitable IDS
IDS Selected
Step 5
Place the IDS in Network Set Up And Customize the IDS as per requirement
IDS Placed & Customized
Check for successful alerting.
Step 6
Complete automation by using alerts.
Automation Achieved
Check New routes in Border Routers
Step 7
Add further functionalities as per Organizational Requirement. (If Desired)

Figure 8. Bottom-up approach for achieving automated alerting for BHR.

C. ANTICIPATED LIFE-CYCLE OF THIS WORK

The research involved in this thesis is primarily intended to safeguard the network (more specifically, the “Autonomous System” (AS) network) on which the target host comes under attack. A collateral benefit of the research is that it will relieve the target host of the D/DoS attack traffic as well, so long as the attack traffic originated from outside of the host’s AS network. As long as D/DoS attacks exist, the BHR solution is an option for implementation by the ISP provider/customer. The technological trend in the network security world—as is the case in the world of traditional warfare—is that, as the technological upper hand is temporarily achieved by the defense, the attacker devises ever more complicated attacks. The crude, but effective, “route-to-null” defense mechanism employed by BHR is so fundamental to how traffic is controlled that this should always provide a high degree of network defense “triage” against what might otherwise be crippling D/DoS traffic volume.

This thesis implements IP version 4, whereas the implementation of IP version 6 in the near future is inevitable. But due to the basic behavior of packet-switched routing, the automated alerting of BHR would be equally effective in an IPv6 environment. So, the life-cycle of this project is as long as D/DoS attacks exist in the network world.

IV. IMPLEMENTATION AND VALIDATION

As illustrated in the bottom-up approach in the previous chapter, this chapter describes the actual implementation of the D/DoS alerting process and the tests used to validate the implementation. The initial sections of this chapter discuss the role of the alerter (IDS), and the process followed by this work in selecting the optimum alerter. Later sections of this chapter discuss the BGP protocol (BGP) and-more specifically-which triggering process was employed to affect the appropriate BGP null-route update.

The final section of the chapter discusses the network setup, the fine-tuning of the selected IDS, the automation of the BHR process, and, finally, the testing of the automated process that was developed.

A. REMOTE (NIDS) VS. CUSTOMER (HIDS) ALERTING

1. What is an Intrusion Detection System (IDS)?

IDSs are systems which are used to detect malicious network traffic followed by taking appropriate action in response. As defined by Wikipedia:

> An Intrusion Detection System generally detects unwanted manipulations to computer systems.[15]

These manipulations are mainly malicious activities, like unauthorized logins; denial of service attacks, malware (which can be in the form of viruses and worms), etc. There are various types of IDSs available on the market and each has its advantages and disadvantages, as discussed in Section B of this chapter. An IDS that is designed to monitor traffic for multiple hosts in the network is known as a network-based IDS (NIDS). Similarly, an IDS that is used to detect changes or malicious activity for one specific host is called a host-based IDS (HIDS). Generally, a NIDS is placed at remote location and can be analogized as a *remote* IDS. On the other hand, an HIDS is a piece of software which is installed on the local machine where attacks and malicious activities are to be detected and hence can be analogized as a *local* IDS.

[15] Wikipedia, http://en.wikipedia.org/wiki/Intrusion_detection_system, last accessed 9 July 2007.

2. Reliability of NIDS vs. HIDS

A NIDS is generally deployed on the network/LAN, whereas a HIDS is deployed on a host. Both have their own pros and cons. Network security professionals working for large corporations or important organizations, where data protection is of utmost importance, face a challenge: Which is the best way to detect an attack and secure all the important hosts within the network. The answer to this question may never be the same in any given scenario. A smaller organization, which has only one kind of client operating system, may prefer a NIDS. On the other hand, bigger organizations, which have multiple operating systems installed on their network, may prefer a HIDS installed on every client or a NIDS installed in conjunction with the HIDS.

A NIDS monitors every packet that is passes through the network. This is sometimes achieved by installing multiple sensors (receiving points) in the network. The various advantages and disadvantages of the NIDS are as follows:

<u>Advantages:</u>

- Very effective when used to detect the known attacks for which a signature is readily identifiable.
- The NIDS runs in stealth mode and makes it hard for the attacker to know of its presence.[16]
- The NIDS is generally not run as an “in-line” device, but rather passively “sniffs” a copy of all packets traversing the monitored network. Hence, a NIDS deployment does not adversely affect normal traffic throughout.
- Multiple sensors can be deployed for one NIDS in large networks.

<u>Disadvantages:</u>

- In larger networks, the NIDS faces the challenge of insertion and evasion techniques that can be used to jumble the traffic associated with one or more or the protected hosts.[17]
- Some less robust NIDS products have been known to crash due to an excessively high volume of traffic, leaving the entire network potentially without any intrusion monitoring protection.

[16] Peddisetty Naga Raju, “State-of-the-Art Intrusion Detection: Technologies, Challenges and Evaluation,” Master’s Thesis, Linkoping, 2005.

[17] Ibid.

- A NIDS will be incapable of scrutinizing header and /or payload information in encrypted packets.
- Placements of NIDS sensors (connection points) is more problematic when working in a switched environment, as most traffic will be unicast vice broadcast unless the switch(es) support the configuration of a port in promiscuous mode

The HIDS monitors every packet passed to the local host and the traffic is passed to the intended application only if there are no malicious packets. HIDS are generally more Operating System (OS)-specific than NIDS. Various advantages and disadvantages of the HIDS are as follows.

Advantages:

- Switch-based networks, which are an issue for NIDS, are not an issue for HIDS.
- An up-to-date HIDS can protect the local host much more effectively than a NIDS.
- An OS-specific attack, which may go unnoticed by a NIDS, is more likely to be detected by a HIDS.
- Any malicious modification of the local machine can be more easily detected by the file-integrity checkers that complement a HIDS.
- The HIDS will be able to scrutinize formerly encrypted traffic as it will process the received packet/payload after its receiving host has decrypted it.

Disadvantages:

- Generally, the HIDS is very difficult to manage since every host is a sensor on its own.
- There is added administrator workload in maintaining one HIDS per host, vice very few-possibly only one-NIDS.
- Unless the HIDS is used in conjunction with the NIDS, an attack targeting the entire network under attack will not likely be recognized by a HIDS.

3. Scalability of Network

The scalability factor plays an influential role while selecting the IDS product and also becomes one of the deciding factors when selecting the NIDS or the HIDS. In a scenario where the organization is using a pure HIDS implementation, additional budget resources should be allocated for new hosts. In the case of a pure NIDS implementation, the NIDS should be able to serve additional hosts. When the NIDS and HIDS are working in conjunction with each other, this depends on the kind of scalability (whether the NIDS itself will be able to serve the client or additional budget resources are required with the HIDS).

B. WHICH IDS BE SELECTED FOR THIS RESEARCH?

HIDS for complete solution and NIDS for a LAN solution.

The above quote is attributed to Ricky Magalhaes, who authored one of the white papers "HIDS vs. NIDS".[18] Bearing in mind that BHR is one of the countermeasures against D/DoS and the consider various aspects discussed in this section, NIDS seems a better choice. Further, since the BHR is not a protective solution for a single, or even selected, host(s) in the network, but rather is a protective solution for all the hosts within the target's organization/network; the network-sensing NIDS appears to be the better alerting tool than a single host-sensing HIDS. This strengthens our point to choose NIDS.

Once the decision had been made that NIDS is the more suitable for our purposes, the next step was to identify the most favorable NIDS product to serve as the BHR alerter. It was surprising to note that IDSs are generally often erroneously understood as firewall type products and only 0.1 % of corporate networks are spending the required budget resources on NIDSs.[19] This strengthens our argument for careful analysis in selecting the optimum NIDS. In our research, importance had been given to the following general features when we selected the optimum NIDS.

- An NIDS which can be customized for organizational requirements.
- Since D/DoS attacks are very sophisticated and most have new patterns, great importance has not been given to detecting various peculiarities or irregularities in the network traffic (for example, what if the attack pattern suddenly changes?). Instead, importance was given so that once the network is under D/DoS attack and the attack is detected, we can then automate the BHR process. Therefore, this survey was carried out to select the NIDS which gives the capability to write our own rules to detect the known D/DoS.

[18] Ricky M. Magalhaes, "Host based vs. Network based IDS," www.windowsecurity.com/articles/Hids_vs_Nids_Part1.html, last accessed 20 July 2007.

[19] Ibid.

- An NIDS which can integrate third-party scripts with the alerting process.
- Software that is freely available.

C. IDS MARKET SURVEY (WHAT'S AVAILABLE)

Much time was spent in identifying suitable NIDSs for this research. The market survey was primarily based on NIDS products. Factors mentioned in Section B above were considered during the survey. It was assumed that by following these factors, if we could successfully detect an attack and then automate the process, special NIDS software can always be developed to meet organizational requirements. Special NIDS software will have the following distinguishing characteristics:

- Ability to detect anomaly-based attacks.
- Ability to recognize new patterns of attacks and take suitable actions.
- Ability to automate BHR process once the alert is generated.
- Unusual events which do not require automation of BHR should still generate alerts.

Designing special NIDS software with the attributes mentioned above (designed for organizational requirement) is a research project of its own. While doing a market survey for this project, I had concentrated on the following traits, along with the general features mentioned in Section B. It makes sense that if the following listed four properties could be achieved, then designing special IDS software with the above features would not only be achievable but would also be very effective for organizational requirements.

- The alerter should be able to recognize the specially crafted attack for this project.
- It should allow us to write customized rules in order to recognize the traits mentioned in Step 1.
- It should have the ability to take automated actions against crafted attacks.
- It should perform normal actions when an attack other than the crafted attack is recognized.

1. Market Survey

A fairly large amount of commercial (with or without single-host free trial) and freeware NIDS products were available in the market. The idea was to select the one which incurs minimum cost and fulfills the maximum desired features. Benefits were obtained from reviewing previous research on NIDS products [12] [13]. An additional market survey was carried out on at least three NIDS products by either contacting the marketing professionals (in the case of commercial product) or the developer (in the case of freeware).

Appendix B presents the results of previously conducted research, as mentioned above, and presents additional information that was obtained by visiting the web site of the product. An additional market survey of a few more NIDS and the optimum NIDS selected is discussed later in this section.

2. Signature- vs. Behavior-Based Detection

It is worth mentioning that no IDS discussion is complete until we discuss the detection method of the IDS. The detection method of any IDS can be broadly categorized into two categories: signature-based and behavior-based detection.

a. Signature-Based Detection

A signature-based IDS is one that detects attacks purely on the basis of signatures, which are either created earlier or installed during setup and then updated later. This kind of NIDS is efficacious in countering known attacks against the network. Known attacks can be in the form of worms, viruses, or attacks created (of known patterns) simply to cause a D/DoS. A famous example of a worm is the release of the Code Red worm in July 2001. Code Red infected nearly 360,000 servers in 14 hours in the second wave of attack.[20] Recent examples of worms include the mass-mailing e-mail worm "Mydoom" that appeared in 2004. Mydoom replicated up to 1000 times per minute and overflowed the Internet with 100 million infected messages in 36 hours.[21] "Melissa," which is an example of spreading an e-mail virus, made use of a Microsoft Word macro embedded in an attachment. All of these highlight the importance of deploying signature-based IDSs in general and our discussion in particular. To detect these harmful activities,

[20] William Stalling, "Network Security Essential, Third Edition," Pearson Prentice Hall, 2007, p. 340, pp. 342-343.

[21] Ibid.

this kind of IDS requires a vast amount of signatures in its database. String/pattern matching is done with the already known patterns to identify the attack.

b. Behavior-Based Detection

Behavior-based detection is also called anomaly-based detection. According to Wikipedia: "An anomaly-based Intrusion Detection System is a system for detecting computer intrusions and misuse by monitoring system activity and classifying it as either normal or anomalous. The classification is based on heuristics or rules, rather than patterns or signatures, and will detect any type of misuse that falls out with normal system operation."[22] This kind of detection is required when organizations do not want to accept any deviation from normal traffic. An anomaly-based IDS raises an alarm when it monitors a deviation from some pre-defined "normal" baseline.

3. Comparison (Advantages/Disadvantages)

Advantages and disadvantages have been limited to the detection methods discussed in the previous section.

Continuing the discussion on signature-based versus anomaly-based IDSs, let us take the scenario that the Code Red worm has just been released. When this kind of scenario happens, there are no existing signatures in the database which will match. This discussed plot outline is called the "zero-day exploit" in the computer security world. The inability to recognize a "zero-day exploit" is the biggest disadvantage for signature-based systems. Another disadvantage of signature-based systems is that even a known attack can be missed if its signature is missing from the database. Therefore, signature-based IDSs demand persistent surveillance by network administrators and the regular/routine updating of signatures.

It is very evident from our above discussion that signature-based systems are less likely to generate false positives and negatives: there either *is*, or *is not* a match on a known attack signature. On the contrary, anomaly-based detection systems generate a

[22] Wikipedia, http://en.wikipedia.org/wiki/Anomaly-based_intrusion_detection_system, last accessed 30 July 2007.

large amount of false alarms (bad) but can be very effective in the detection of zero-day exploits (good). Anomaly-based detectors are very helpful for the network security community in recognizing new patterns of attack.

4. Detection Method for Our Research

The basic, and most important, purpose of this project was to explore the effectiveness of automation of BHR and the advantages achieved from its implementation. Although the above factors play a vital role in the selection process, I was not arduous in selecting the detection method for the selected NIDS. If the alerter provided the capability for defining one's own rules (for example, setting thresholds to detect ICMP flooding attacks) and had the capability to trigger on that rule, then that product was considered suitable for my purposes.

5. Optimum Alerter Selection

Before we conclude our discussion on which alerter we selected for our project, the following paragraphs discuss the additional market survey I conducted.

a. Additional Market Survey

In addition to the survey conducted in Appendix B, the following NIDS products were also surveyed:

(1) Bro. The Bro IDS was considered for this survey because it is open-source and freely available for download. As part of the survey, I posed a few questions to Mr. Vern Paxson, who is actively involved in the Bro research project[23]. His answers indicated that it would be very complex to integrate Bro into our research. Questions asked and replies received via e-mail are attached as Appendix C. In addition, Bro uses a specialized policy language which would have been an additional task to understand. Therefore, Bro was not chosen for this project.

[23] Lawrence Berkeley National Laboratory, http://www.bro-ids.org/, last accessed 13 September 2007.

(2) Splunk. Splunk is another product that is available in the NIDS market. Splunk, in reality, is a search engine, which gives the flexibility to search, access logs and configurations, write one's own scripts, etc. Splunk is a commercial product but a free download for testing for a single user is available online.

The biggest disadvantage of this product (which made it not feasible for this research) is that it cannot be used as a real-time automated solution. Questions posed to a Splunk sales engineer brought me to the conclusion that this product couldnot be used for our research because Splunk would introduce a large delay in the triggering mechanism. Introducing an unnecessary delay would defeat the very purpose of automating BHR. Questions posed to the Splunk engineer are attached as Appendix D.

b. Optimum Alerter Selected

After a thorough survey of eight products available on the market, Snort , originally developed by Martin Roesch,[24] was selected as undoubtedly the optimum alerter due to following reasons:

- "Snort® is an open source network intrusion prevention and detection system utilizing a rule-driven language, which combines the benefits of signature, protocol and anomaly based inspection methods."[25]
- With over 3 million downloads and 100,000+ active users, Snort has undoubtedly become the most popular freeware IDS deployed around the world.[26]
- Due to its wide popularity and deployment, Snort forums and communities often offer immediate solutions to problems encountered during deployment and after deployment.
- Snort can be integrated to hundreds of third-party solutions.
- Snort is highly customizable as per organizational requirement.

[24] Snort.org, http://www.snort.org/training/, last accessed 13 September 2007.

[25] Snort.org, http://www.snort.org, last accessed 30 July 2007.

[26] Snort.org, http://www.snort.org/community, last accessed 30 July 2007.

- Although Snort is highly popular due to its signature-based detection, its available flexibility for anomaly-based inspection methods made it very suitable for our research. Snort allows writing threshold-based rules, which presents favorable circumstances within our project to detect the crafted D/DoS attacks.

D. ALERT NOTIFICATION OF TRIGGER ROUTER

This section discusses in detail the protocol used to implement the BHR route update message format, which was required once the attack was recognized; automation techniques; and, finally, the communication channel between the alerter and the trigger router.

It is good to understand the classic definition of Autonomous System (AS) before steering our discussion to intra-AS and inter-AS protocols. As per RFC 1268, "the classic definition of an Autonomous System is a set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets within the AS, and using an exterior gateway protocol to route packets to other AS's. From the standpoint of exterior routing, an AS can be viewed as monolithic: networks within an AS must maintain connectivity via intra-AS paths."[27]

1. The BGP Protocol

Two routing protocols are used extensively for routing within an AS. These protocols are Routing Information Protocol (RIP) and Open Shortest Path First (OSPF). RIP is a distance vector protocol which is generally implemented in smaller ASs. On the other hand, OSPF, which is a link-state protocol, is implemented in the larger ASs. With OSPF, router broadcasts route information to all other routers within their AS. Routers running RIP broadcasts the routing information only to neighboring within their AS.

Once routers learn all the routes within their AS, the next task is to learn what exists outside their AS. This job is done by the Border Gateway Protocol (BGP). BGP is the de facto standard for inter-AS routing protocols. BGP is generally referred as an exterior gateway protocol. It performs inter-AS routing by exchanging the routing

[27] RFC 1268, www.ietf.org, last accessed 1 August 2007.

information between pairs of border routers serving two separate ASes. This protocol needs to establish a TCP connection using port 179 between the two BGP peering routers for exchanging BGP routing updates.

BGP is used to learn the reachability information from neighboring ASes. BGP also ensures the propagation of the reachability information to the internal routers of an AS. BGP is a complex protocol and has an enormous number of features to implement policy-based decisions to ensure propagation of reachability.

A BGP session between two BGP peers is said to be an external BGP (eBGP) session if the BGP peers are in different ASs. Similarly, a BGP session between two BGP peers is said to be an internal BGP (iBGP) session if the BGP peers are in the same AS. A TCP connection is required to be established in both cases.

2. The iBGP Protocol

iBGP works very similarly to how eBGP works. The difference lies only in how BGP is configured on the router and network boundaries. A primary result of this difference is that iBGP does not advertise outside the AS. Figure 9 represents pictorially the working difference between eBGP and iBGP. Figure 10 and Figure 11 as well as Table 1 and Table 2, portray where the difference lies while configuring the routers.

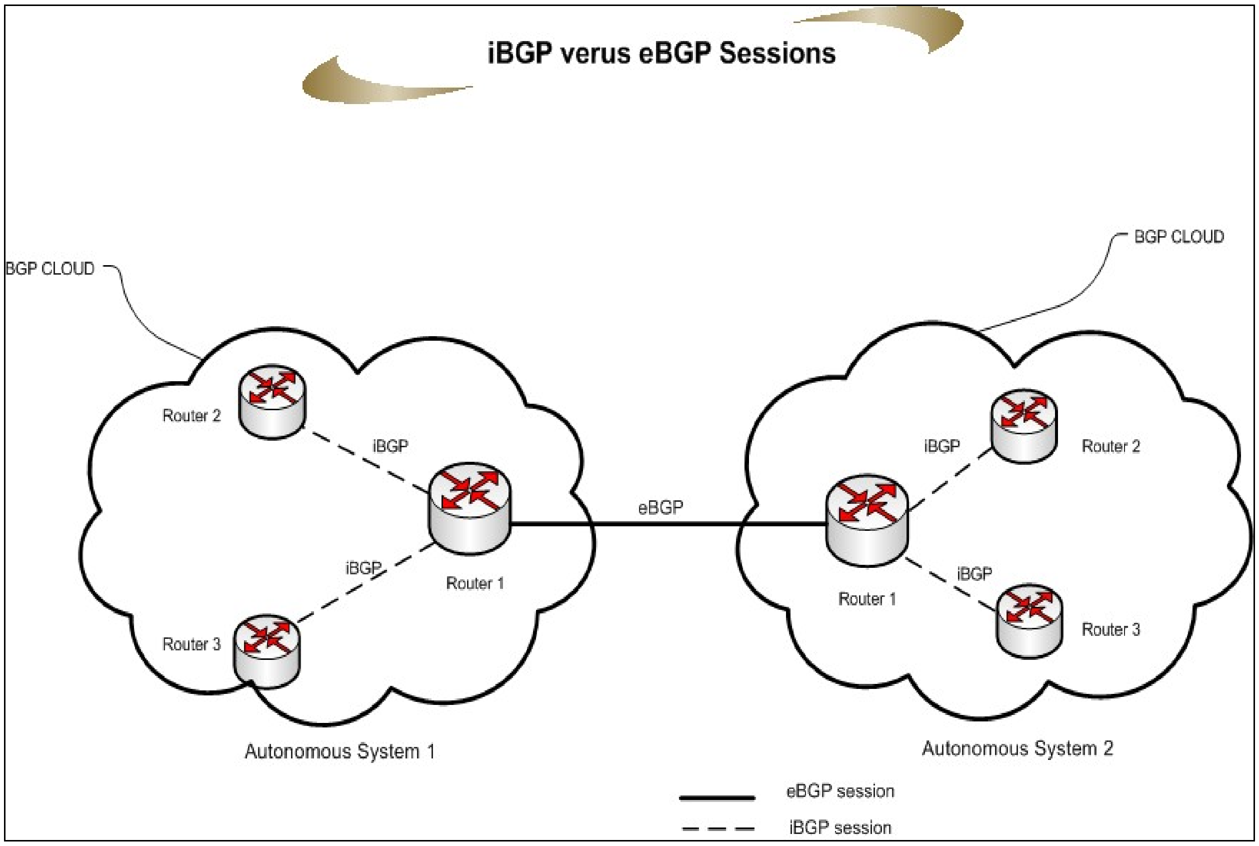


Figure 9. Pictorial difference between iBGP and eBGP session.

In the above figure, a TCP session is established between BGP peer router 1 of AS1 and BGP peer router 1 of AS2. Since the BGP session spans two ASes, this TCP session is an eBGP session.

The session established between router 1 and router 2 as well as between router 1 and router 3 in AS1 is within the same AS and hence it is an iBGP session.

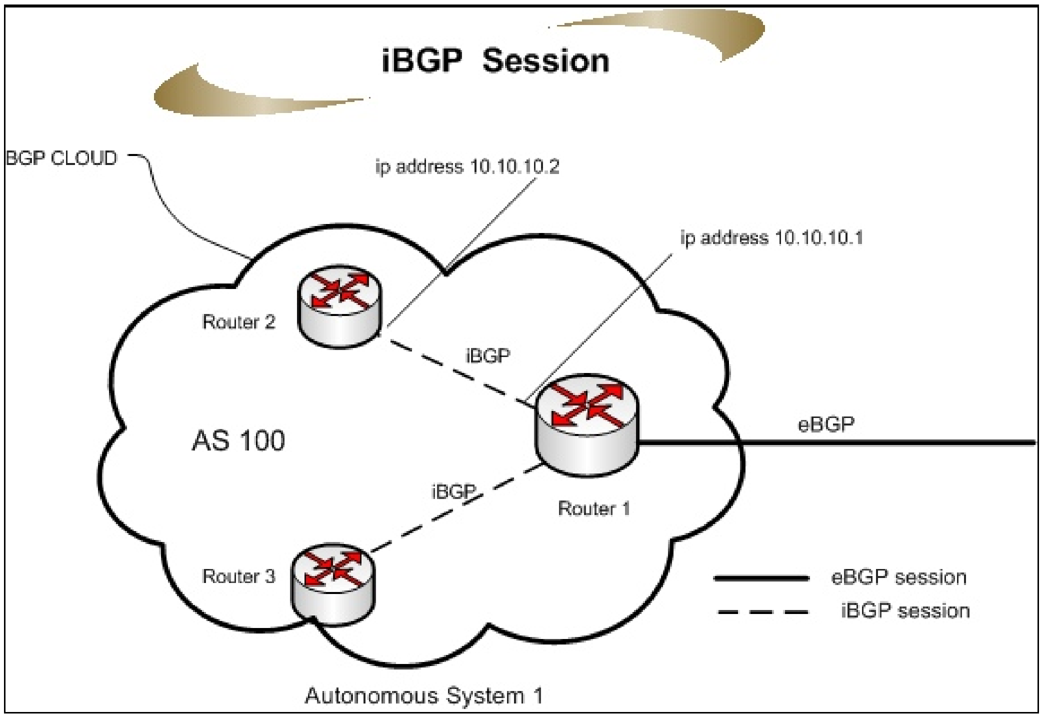


Figure 10. iBGP session.

Every AS in the Internet is allotted a unique number 16-bit integer number, called AS number, which is used for configuring routers. This number is used for exchanging routing information and as an identifier of the AS itself. In our example, suppose the AS number of AS 1 is 100 and 200 for AS 2. The basic configuration of iBGP and eBGP for the example network is shown in Table 1 and Table 2, respectively.

Router 1-AS1	Router 2-AS1

`interface eth xx`
`ip address 10.10.10.1 255.255.255.0`
`!`
 	
`!`
`router bgp **100**`
` neighbor 10.10.10.2 remote-as 100`
	`interface eth xx`
	`ip address 10.10.10.2 255.255.255.0`
	`!`
	
	`!`
	`router bgp **100**`
	` neighbor 10.10.10.1 remote-as 100`

Table 1. Basic configuration of iBGP session between Router 1 and Router 2 of AS 1.

As shown, the eBGP session in Figure 11 and the eBGP configuration in Table 2, note that both the routers belong to different ASs.

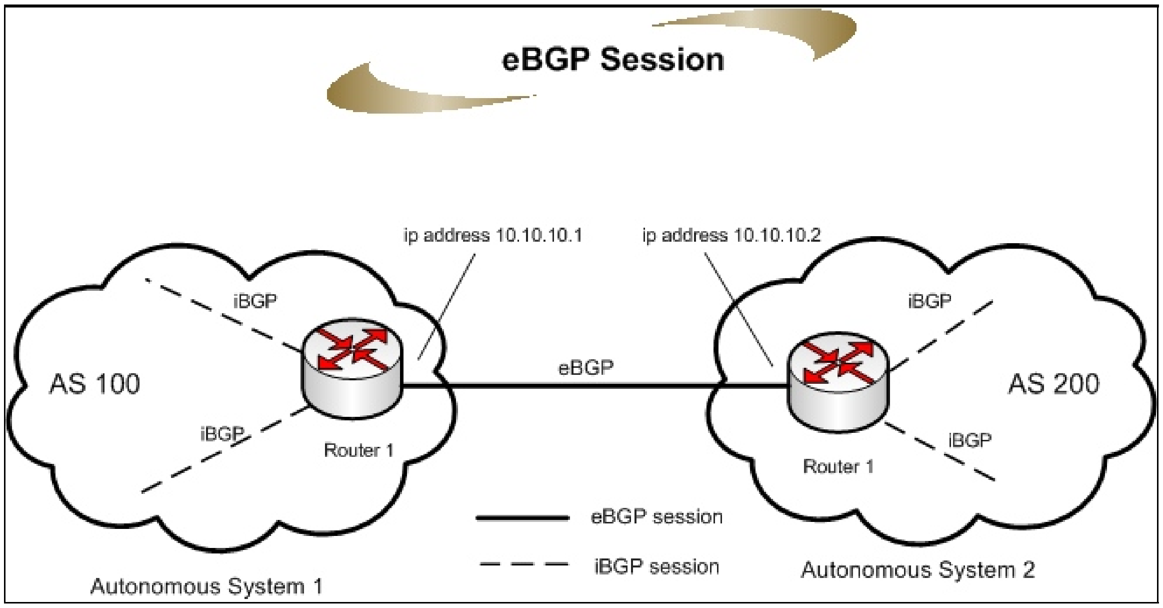


Figure 11. eBGP session.

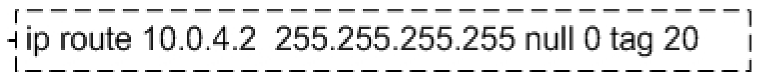
Router 1-AS1	Router 1-AS2

`interface eth xx`
`ip address 10.10.10.1 255.255.255.0`
 	
`!`
`router bgp` **`100`**
`neighbor 10.10.10.2 remote-as 200`
	`interface eth xx`
	`ip address 10.10.10.2 255.255.255.0`
	`!`
	`router bgp` **`200`**
	`neighbor 10.10.10.1 remote-as 100`

Table 2. Basic configuration of eBGP session.

E. ROUTE UPDATE MESSAGE

Upon recognition of an attack, a major responsibility of the alerter was to connect to the trigger router and issue route update commands to add static routes. The IP address that is to be black-holed was set for null routing. The trigger router then used the iBGP route update to propagate this route to all the edge routers (iBGP peers). The iBGP peers then set their next hop to the destination based on this update from the trigger. The route update command for adding a black-hole route is shown in Figure 12.



```
ip route 10.0.4.2  255.255.255.255 null 0 tag 20
```

Figure 12. Route update message.

A detailed explanation of this update command and trigger router configuration is in Appendix A.

1. How to Achieve Automation/Scripting of Transmitted Alert

At this stage, we know that to achieve automation, we needed to find a way to log into the trigger router without human (administrator) intervention. After a successful login process, the next major requirement was to upload the route update command (as shown in Figure 12) into the trigger router.

There are many ways that this can be accomplished and the idea is to achieve this automation in the fastest possible way. In this section, we will discuss a few techniques by which automation can be achieved. The automation method chosen for our research will be discussed in depth when we discuss instrumentation in a late section of this chapter.

A few alternative options for achieving automation for BHR are discussed below. Every approach has its own advantages.

a. Plug-in

Wikipedia defines a plug-in as “a computer program that interacts with a host application to provide a certain, usually very specific, function “on demand.” Applications support plug-ins for many reasons. Some of the main reasons include:

enabling third-party developers to create capabilities to extend an application, reducing the size of an application, and separating source code from the application because of incompatible software licenses."[28]

In our environment, a plug-in was one of the best options. We only needed the alerter to perform certain actions upon recognition of a D/DoS. A plug-in should be able to take care of the login process followed by updating the router with the route update command. A plug-in is also sometimes referred to as an *add-in* or *add-on*. When it comes to performance, the plug-in is faster because these are dedicated to perform very specific functionality.

b. Client-Server Model

This is another good option for achieving automation. Generally, a client-server model can be achieved by implementing a socket mechanism. In this model, the server process waits for the incoming connection and the client process is the one which is user-driven. The server continuously listens to the requests made by clients. On receipt of a request from a client, a new client specific data socket is created by the server and, once the process is complete, the socket is closed.

In the Unix world, implementation of client-server processes this way is referred as daemon processes.[29] In our research, this could be implemented as follows. When the D/DoS attacks are recognized by the alerter, it initiates the client process. On the other end, the server process, which is continually listening for requests, opens the new socket on receipt of a request from a client process. After creating a socket, the server logs into the trigger router and delivers the appropriate null route update. The advantage of this method is that the server process can run on the same machine (i.e., on the alerter) or on a separate machine (in a network where IDSs have more than one sensor).

[28] Wikipedia, http://en.wikipedia.org/wiki/Plugin, last accessed 2 August 2007.

[29] University of Wolverhampton, School of Computing and IT, www.scit.wlv.ac.uk/~jphb/comms/sockets.html, last accessed 2 August 2007.

A few of the functions which are commonly invoked to achieve the purpose are shown in Table 3.

`socket()`	To create a socket
`bind()`	To associate a socket with a network address
`connect()`	To connect a socket to a remote network address
`listen()`	To wait for incoming connection attempts
`accept()`	To accept incoming connection attempts

Table 3. Common functions of socket programming.[30]

c. Search and Destroy

This technique can also be employed to achieve our aim of automation. Most of the IDSs maintain their alert database when the rules are fired on the basis of signature detection, anomaly-based detection, or protocol-based detection. In this technique, the scheme is to search the pattern for which BHR automation is supposed to be triggered. Once the pattern is found in the database, a third-party script can be run to log into the router and deliver the null route update command. For example, we could log the alerts in the SQL database. Then, whenever a new alert is generated, the database is searched and, if the matching pattern is found, a Java script program can be used to log into the trigger router to deliver the route update command. The Java script may invoke another third-party script. For example, the “Expect”[31] software package offers scripts which automate the remote login into other devices by mechanizing telnet, etc.

[30] University of Wolverhampton, School of Computing and IT, www.scit.wlv.ac.uk/~jphb/comms/sockets.html, last accessed 2 August 2007.

[31] The Expect Home Page, http://expect.nist.gov/, last accessed 13 September 2007.

The biggest disadvantage of this technique is delay. Searching might require a long time to match the pattern, especially in a scenario where the alerter is already overwhelmed with an enormous amount of packets during D/DoS attacks.

2. Alerter-to-Trigger-Router Channel

This section discusses the communication channel adopted to achieve the automated triggering process. It basically depends on the network setup. Irrespective of anything else, the alerter is always going to be in the local network. If a customer-based BHR is planned, then the alerter and trigger router are going to be placed in the same network. Direct Ethernet connectivity is the best option. In case, the trigger router is placed at the ISP level; then the communication channel can be agreed upon between the organization and the ISP. The main concern will be security. The automated login should be secured, e.g., SSH (secure shell) should be preferred over telnet because SSH is a network protocol which allows data transfer over a network in a secure fashion.

In our scenario, since we had implemented customer-based BHR, the alerter as well as the trigger router are co-located and thus telnet session is less of a risk than if the traffic was traveling outside the alerter's network.

3. Advertisement Channel

Once the D/DoS attack against the target host is ascertained; how quickly the attack can be halted within the autonomous boundary depends on how fast the trigger router can advertise a black-hole route. The advertisement channel is the communication channel used by the trigger router to advertise new route information to border routers.

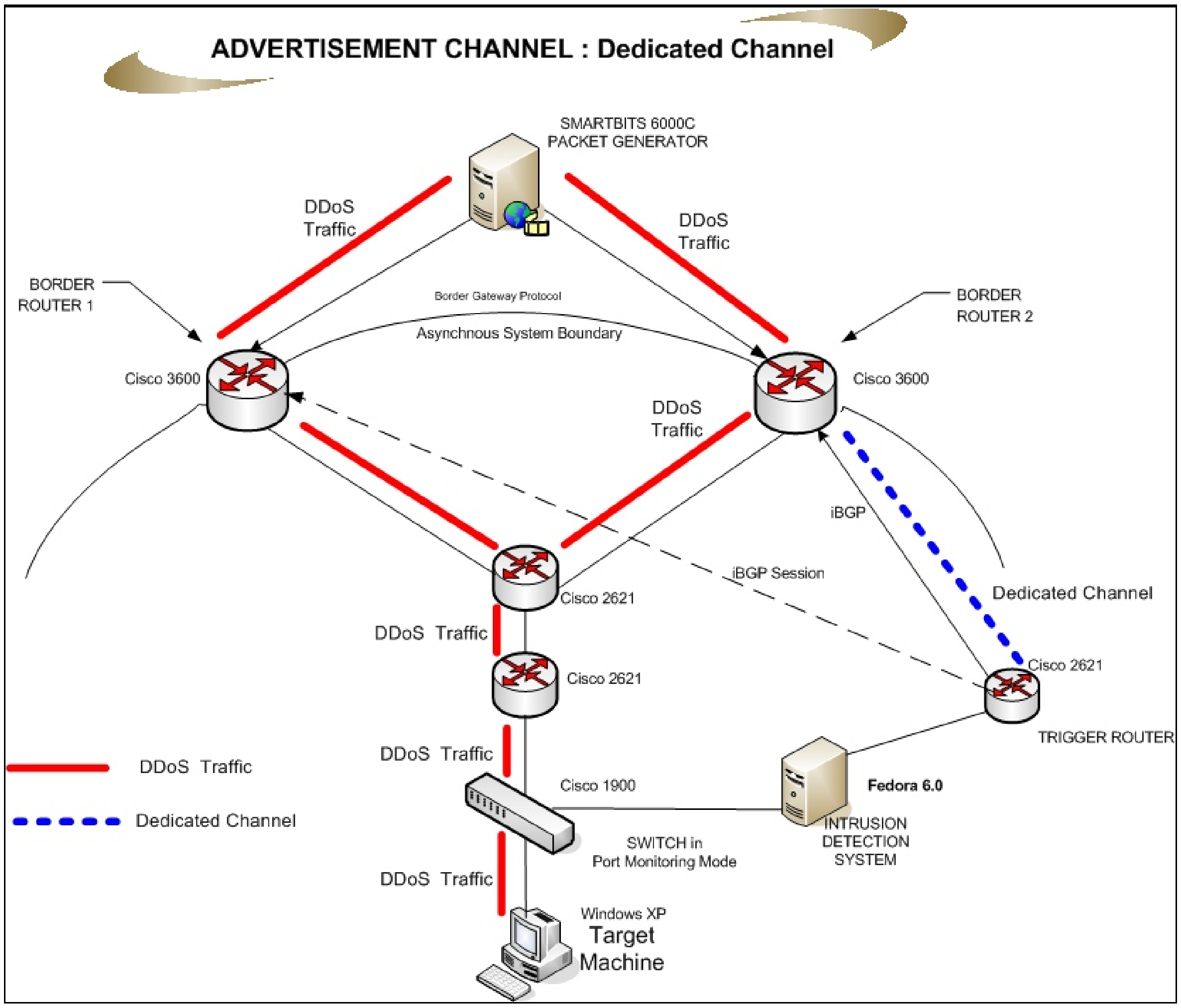


Figure 13. Dedicated channel used for advertisement.

During D/DoS attacks, the existing route from the border routers to the victim machine is already overwhelmed with an enormous amount of packets. In such circumstances, it may be difficult to propagate new routing information through the same channel from where the D/DoS occurred. There are a couple of ways this issue can be resolved; either by having a dedicated channel to the ISP, e.g., modem connectivity, or by reserving some bandwidth on the existing route. Figure 13 and Figure 14 portray both of these scenarios.

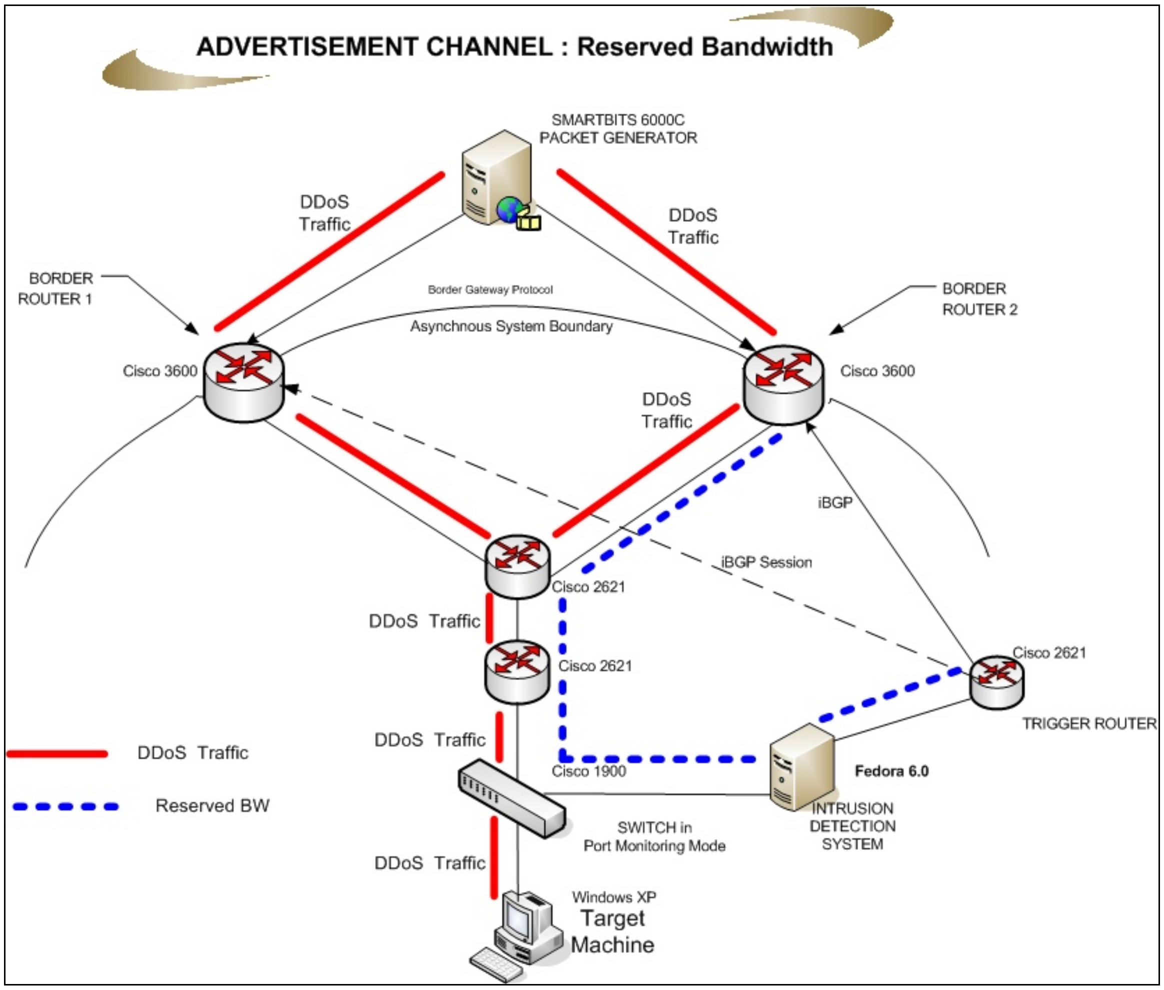


Figure 14. Advertisement by reserved bandwidth (effective when the attack is coming via the same route).

It is essential to prioritize this channel. Using a dedicated channel achieves the BHR automation fastest because this methodology uses a channel which is not affected by the D/DoS attack. It is worth mentioning at this juncture that adopting this method by every customer/organization is not feasible due to the following:

- ISPs cannot provide the direct access to their routers.
- Procuring dedicated channels presents additional cost to any organization.

In our research we used a dedicated channel (Ethernet connection) to one of the border routers to advertise the new route. This simulates the scenario in which there was an agreement for the customer to have access to the ISP’s border router.

F. TEST NETWORK CONFIGURATION

This section describes, the most critical part of the research, including the hardware and software requirements for the test network, the topology and IP routing scheme used, and fine tuning of the selected alerter. Most of the time was spent in configuring the network (routers, switches, etc.), selecting and fine tuning the alerter (Snort), and configuring and modifying a third-party Snort plug-in for automating the BHR.

1. Main Components and Their Configuration

The list of components used (along with the software used) selected for our topology are mentioned below. Later in the section, the configuration for each component is also discussed.

a. Hardware and Software

The following devices were used for our test bed network:

- One Cisco router with IOS C2600 software, Version 12.1(14), with five 10 Mbps Ethernet Interfaces, used as an internal router.
- One Cisco router with IOS C2600 software, Version 12.1(13), with five 10 Mbps Ethernet Interfaces, used as an internal router.
- One Cisco router with IOS C2600 software, Version 12.0(14), with five 10 Mbps Ethernet Interfaces, used as a trigger router.
- Two Cisco routers with IOS 3600 software, Version 12.2(3) with four 10 Mbps Ethernet Interfaces and one 100 Mbps Fast Ethernet Interface.These routers were used as Border Routers.
- Two Cisco 1900 16-port switches in the port-forwarding mode.
- Two desktop PCs with Windows XP SP 2. One was used as a target machine and the other was used as an attack monitor. The attack monitor was placed outside the AS boundary.
- One desktop PC with Fedora 6.0 loaded on it for the IDS.
- One Smart Bits 6000C Performance Analysis System of Spirent Communications for continuous traffic generation.

- One LAN-3321A TeraMetrics XD module with two 10/100/1000 Mbps Ethernet Copper ports and two 1 Gigabit Ethernet Fiber ports installed on the Smart Bits 6000C system. Both the copper ports were used to simulate a D/DoS attack.

The applications used for our research were as follows:

- Smart Window version 7.70.128, for use with the Smart Bits 6000C system.
- CommView version 5.5 of Tamosoft, for crafting custom packets.
- Ethereal version 0.10.7 (C) running with WinPcap (3.0 alpha3) for capturing traffic.
- Wireshark version 0.99.3a Network Protocol Analyzer with libpcap version 0.9.4 on Linux 2.6.18-1.2798.fc6.

Arrangement of all the constituent elements of the network as well as the IP routing scheme followed to design the network in our scenario is depicted in Figure 15.

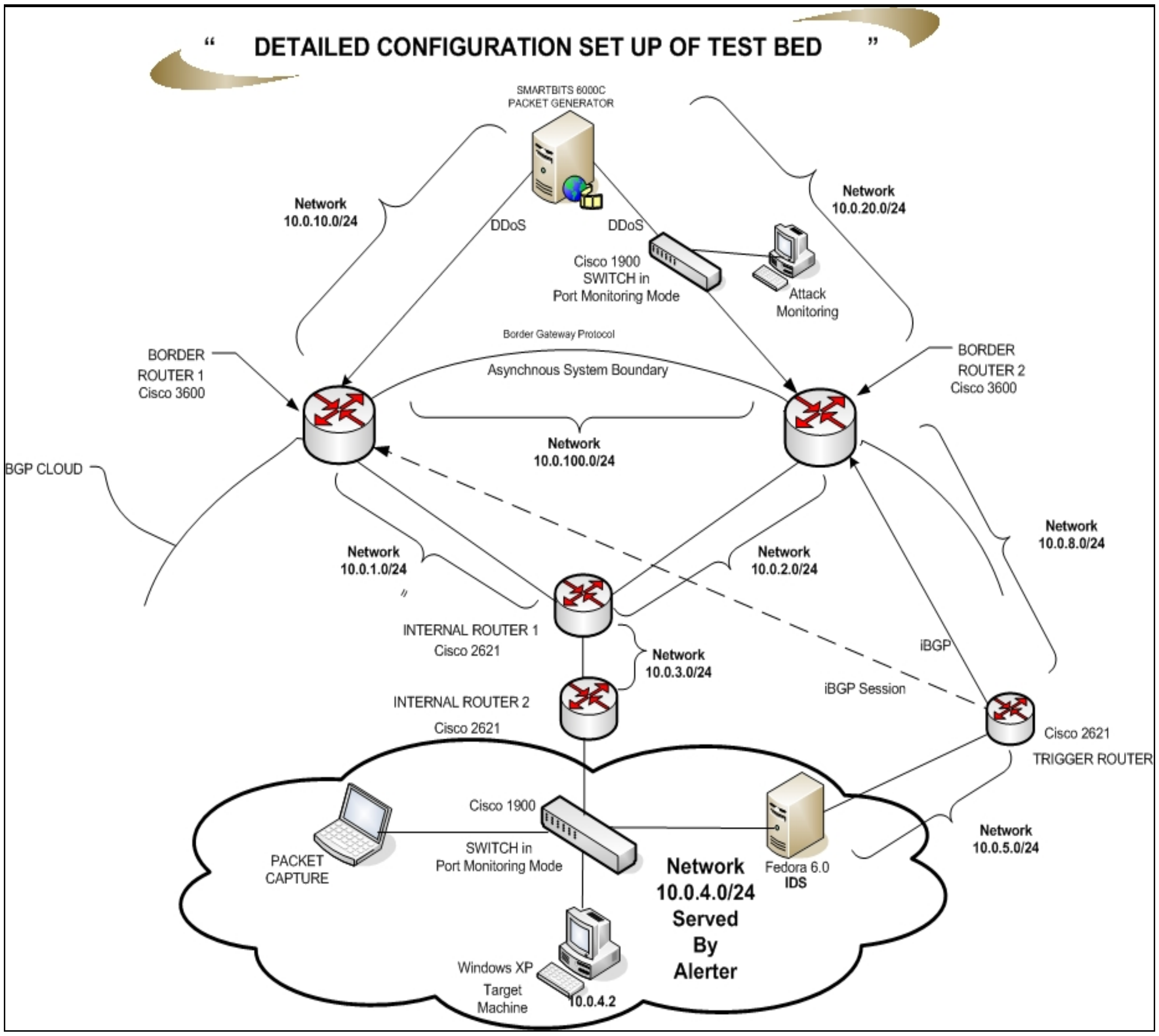

"DETAILED CONFIGURATION SET UP OF TEST BED"
SMARTBITS 6000C PACKET GENERATOR
Network 10.0.10.0/24
Network 10.0.20.0/24
DDoS
DDoS
Cisco 1900 SWITCH in Port Monitoring Mode
Attack Monitoring
Border Gateway Protocol
Asynchnous System Boundary
BORDER ROUTER 1 Cisco 3600
BORDER ROUTER 2 Cisco 3600
Network 10.0.100.0/24
BGP CLOUD
Network 10.0.1.0/24
Network 10.0.2.0/24
Network 10.0.8.0/24
INTERNAL ROUTER 1 Cisco 2621
Network 10.0.3.0/24
iBGP
iBGP Session
INTERNAL ROUTER 2 Cisco 2621
Cisco 2621 TRIGGER ROUTER
Cisco 1900
SWITCH in Port Monitoring Mode
PACKET CAPTURE
Network 10.0.4.0/24 Served By Alerter
Fedora 6.0 IDS
Network 10.0.5.0/24
Windows XP Target Machine
10.0.4.2

Figure 15. Detailed network setup with IP scheme.

b. Routers (Border, Internal, Trigger)

As shown in Figure 15, five routers were utilized to form the test bed. Two routers were configured as border routers to simulate the AS environment. Interfaces of the border routers which are facing the internal routers are configured with the OSPF protocol. Two internal routers were configured with only the OSPF protocol to simulate the environment of the network within the AS. Sample configurations of the border routers and internal routers are provided in Appendices E and F, respectively.

c. Target

For the sake of simplicity, one Windows XP machine was selected as a target machine. The IP address 10.0.4.2 was assigned to this Windows XP machine, as shown in Figure 15. Ethereal was loaded onto this machine to capture the packets and to note the efficacy of the D/DoS attack. Further, to see the effect of D/DoS attacks, network utilization was also monitored, which is discussed in next section.

d. Alerter (IDS)

This was the most important and critical component of the network setup. After selecting the optimum alerter, as discussed in Section B of this chapter, setting up and fine tuning this crucial element of the network, took most of the allotted time.

Snort version 2.6.1 software was downloaded from the official Snort site. Sourcefire reference manuals were very helpful in setting up the alerter as per our environment.[32] [33] Snort was installed on Fedora 6.0. Snort software versions are also available for Windows, Solaris, etc. Research revealed that Snort is most stable with a Linux-based environment. The following are the step-by-step details followed for setting up Snort.

2. Setup of Snort

There are various ways Snort can be installed: by downloading the RPM directly or from the source code. The approach I followed was a mixed approach. Along with installing Snort, a graphical interface called BASE was also installed. BASE was installed in order to verify Snort’s capability as an “attack recognizer” and an “alert notifier” once an attack was recognized.

a. Step 1

Before performing Snort installation, we needed to perform some basic network settings and configure the services we needed to run. During our setup, the

[32] Sourcefire, “Building and Operating Snort,” Security Training Program, 2004-2007.

[33] Sourcefire, “Snort Rules,” Security Training Program, 2004-2007.

firewall was turned off for simplicity. The firewall can be part of a hardening exercise once the automation was achieved. While configuring the network, the following should be configured without any ambiguity:

(1) IP Address. Two IP addresses were allocated to alerter. One IP address was configured for Ethernet which sniffs the traffic for network 10.0.4.0/24. This IP address need not belong to a specific network because this interface runs in promiscuous mode. As shown in Figure 15, the other interface is part of the 10.0.5.0/24 network. Therefore, IP address 10.0.5.2 was allocated to the other Ethernet port, which was connected to the trigger router.

(2) Netmask. A network mask of 255.255.255.0(/24) was used.

(3) Gateway. Although in our scenario it was not mandatory, it is a good practice to provide a default gateway address.

(4) DNS Server. In our scenario, we did not configured a DNS server.

(5) Services. Ports 22, 80, 443, and 3306 were enabled to support SSH, HTTP SSL and MySQL services in the Fedora box. If the MySQL package is not installed in Fedora, then the following packages need to be installed from the Fedora 6.0 CD or by downloading the RPMs from the Internet. In order to achieve a trouble-free configuration setup, the following MySQL packages should be installed:

mysql

mysql-bench

mysqlclient-devel

mysql-connector-odbc

mysql-server

b. Step 2

Before installing Snort, several components were required to be pre-installed. The following components were downloaded from the Internet as source code and then **downloaded** into the /usr/local/src directory:[34] [35]

```
pcre-6.3.tar.gz

libnet-1.0.2a.tar.gz

libpcap-0.9.4.tar.gz
```

The packages above allowed using Snort rules, tearing down TCP sessions in response to an alert, and putting the interfaces into promiscuous mode to read network traffic off the wire.

These packages were **extracted and complied** in the /usr/local directory by giving the following commands in sequence for every package:

```
tar zxvf src/pcre-6.3.tar.gz

cd pcre-6.3

./configure

make

make install
```

The first step shown above extracted the PCRE package. The second step changed the /usr/local to the /usr/local/pcre-6.3 directory. Steps three through five compiled the PCRE package. A similar process was followed for LIBNET and LIBCAP packages.

c. Step 3

At this stage, we installed the Snort and configured the MySQL database to configure the Snort alerts. This step also included; creating a few directories that would be used by Snort.

[34] SoureForge, Inc., http://sourceforge.net/, last accessed 3 September 2007.

[35] tcpdump/libpcap, http://www.tcpdump.org/release/, last accessed 3 September 2007.

Snort version 2.6.1 was downloaded into /usr/local/src directory. The file name of the Snort package is snort-2.6.1.3.tar.gz. This file was extracted and compiled as follows.

```
tar zxvf src/snort/snort-2.6.1.3.tar.gz

cd snort-2.6.1.3

./configure   - -with –mysql

make all

make install
```

The series of commands listed above successfully installed Snort in the Fedora machine. The third command indicated that Snort was compiled with MySQL.

Snort’s official website also provided the up-to-date rules, which were downloaded into the /usr/local/src directory and copied into newly created directories of Snort as follows.

```
mkdir  /etc/snort

mkdir /var/log/snort

tar zxvf /usr/localsrc/snortrules-snapshot-Current.tar.gz –C /etc/snort

cp etc/*.conf*   /etc/snort

cp etc/*.map     /etc/snort

ln –s    /usr/local/bin/bin/snort  /usr/sbin/snort
```

The six commands above were given in the /usr/local/snort-2.6.1.3 directory. The following three commands created a Snort user and user group in Snort directory.

```
groupadd snort

useradd  -g snort  snort

chown snort:snort  /var/log/snort
```

At this stage, a few configuration changes were required in order to get Snort up and running. Configuration changes were made in a file called snort.conf, which existed within the /etc/snort/snort-2.6.1.3 directory. We edited this file with the vi vi editor by giving the following command.

```
vi  /etc/snort/snort.conf
```

Once the file is opened, string " var RULE_PATH" was searched. Make the modification to the variable as follows:

```
var  RULE_PATH   /etc/snort/rules
```

After making the above modification, we searched for the following string "database: log to variety of databases" and then added the following line at a point directly after the commented lines.

```
output database:  log, mysql,  user=snort   password=password dbname=snort
host=localhost
```

The line above tells Snort to log the events in the MySQL database. Snort was also provided with the details of the database. The database name was "snort", the user name was also "snort", and the password was "password".

At this point, Snort was installed but, in order to check the working of snort, we needed to create the database named "snort" in MySQL. To achieve this, issue the following statements:

```
mysql
SET PASSWORD FOR root@localhost=PASSWORD('password');
create database snort;
grant CREATE, INSERT, SELECT, DELETE, UPDATE on snort.* to
snort@localhost;
SET PASSWORD FOR snort@localhost=PASSWORD('password')
exit
```

The above statements created a database named snort, in which Snort would log its events. The Snort package also contained the schema for various databases. These schemas were located in the snort-2.6.1.3 directory. The following commands activated the database schema.

```
/usr/local/snort-2.6.1.3/schemas
mysql –p < create_mysql snort
```

At this stage, the database called snort was created, which can be tested by giving the following commands:

```
mysql –p
(Enter password, which is “password” as per our configuration.)
SHOW DATABASES;
```

Now, we could test the Snort installation by giving the following command:

```
/usr/local/bin/snort -c /etc/snort/snort.conf
```

The Snort process created the alert file under /var/log/snort/ on its own. We needed to change the permissions of the alert file so that the Snort user could access that file. This was achieved by giving the following commands.

```
chown snort: snort   /var/log/snort/alert
chmod 600 /var/log/snort/alert
```

d. Step 4

In this step, we installed BASE and ADODB packages. The ADODB package provided the interface between the GUI and the MySQL database. The BASE package provided the graphical front end to the Snort database. These packages were downloaded from sourceforge into the /usr/local/src directory.[36] These packages helped

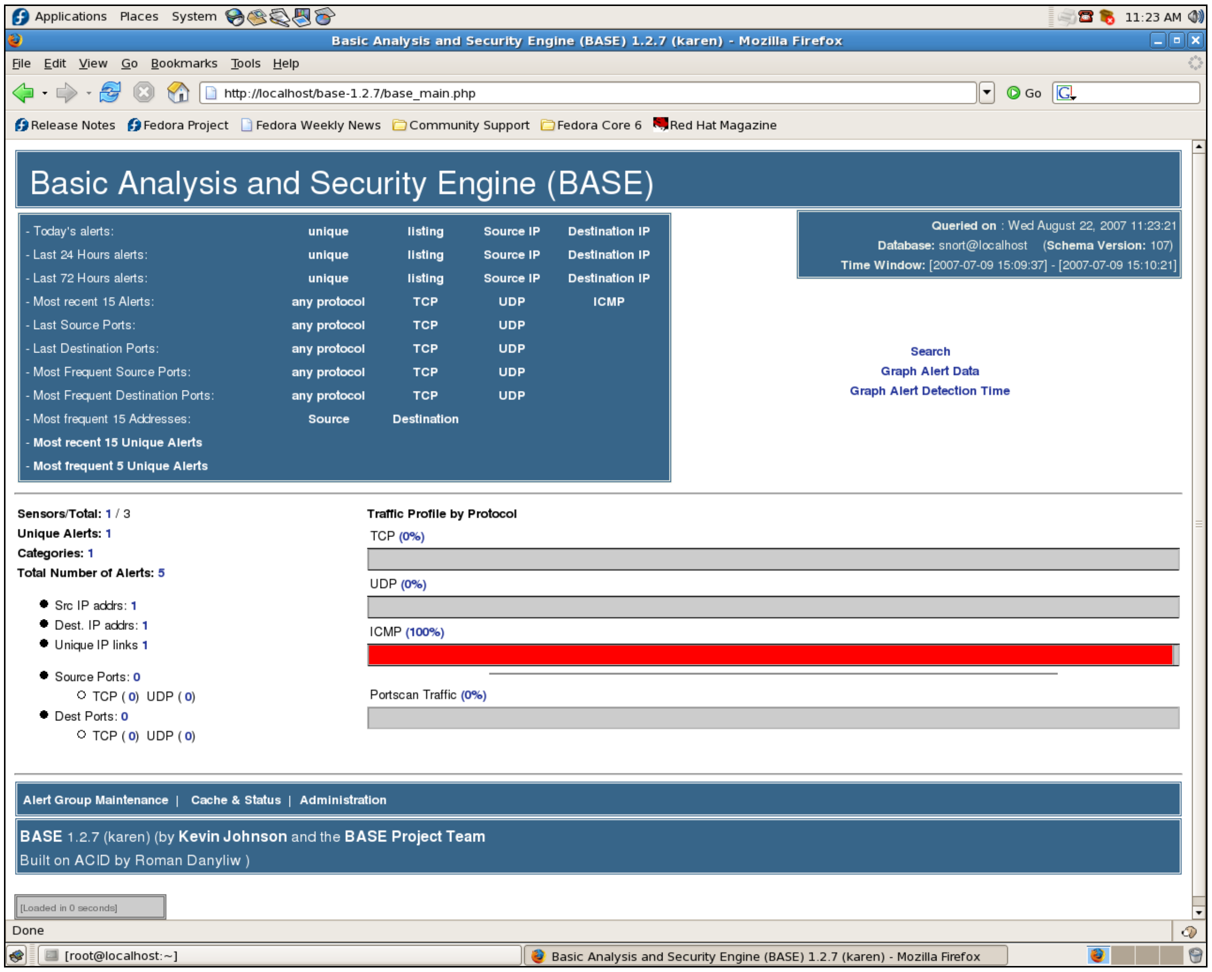
[36] SoureForge, Inc., http://sourceforge.net/ last accessed 5 September 2007.

in viewing alerts in an organized fashion. These were installed to ensure the proper functioning of Snort and its customized Snort rule.

```
cd  /var/www/html

tar zxvf /usr/local/src/adodb490.tgz

tar  zxvf /usr/local/src/base-1.2.7.tar.gz

chown apache base-1.2.7

service httpd restart
```

Now, since we restarted the http service, we configured the BASE by opening the browser. We opened the browser with URL http://localhost/base-1.2.7.

It was a fairly simple process. Once the above URL was typed, the BASE setup program started on its own. It prompted for the path to ADODB in the first step. The path name was given as /var/www/html/adodb. The next step was to enter the database name, database host, database user name, and database password. Enter those details exactly as configured in Step 3 of this section. We submitted the query by clicking “the submit query” button on the screen. We followed the on-screen instructions in the setup script to create the database tables used by the BASE application. Clicking the “Create BASE AG” button created tables. Now, the login screen was presented. Upon providing the appropriate login credentials, the BASE main screen looked like the following:


Applications Places System
11:23 AM
Basic Analysis and Security Engine (BASE) 1.2.7 (karen) - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/base-1.2.7/base_main.php
Go
Release Notes Fedora Project Fedora Weekly News Community Support Fedora Core 6 Red Hat Magazine
Done
[root@localhost:~]
Basic Analysis and Security Engine (BASE) 1.2.7 (karen) - Mozilla Firefox

Basic Analysis and Security Engine (BASE)

- Today's alerts:	unique	listing	Source IP	Destination IP
- Last 24 Hours alerts:	unique	listing	Source IP	Destination IP
- Last 72 Hours alerts:	unique	listing	Source IP	Destination IP
- Most recent 15 Alerts:	any protocol	TCP	UDP	ICMP
- Last Source Ports:	any protocol	TCP	UDP	
- Last Destination Ports:	any protocol	TCP	UDP	
- Most Frequent Source Ports:	any protocol	TCP	UDP	
- Most Frequent Destination Ports:	any protocol	TCP	UDP	
- Most frequent 15 Addresses:	Source	Destination		
- Most recent 15 Unique Alerts				
- Most frequent 5 Unique Alerts				

Queried on : Wed August 22, 2007 11:23:21
Database: snort@localhost (**Schema Version:** 107)
Time Window: [2007-07-09 15:09:37] - [2007-07-09 15:10:21]

Search
Graph Alert Data
Graph Alert Detection Time

Sensors/Total: 1 / 3
Unique Alerts: 1
Categories: 1
Total Number of Alerts: 5

- Src IP addrs: 1
- Dest. IP addrs: 1
- Unique IP links 1

- Source Ports: 0
  - TCP (0) UDP (0)
- Dest Ports: 0
  - TCP (0) UDP (0)

Traffic Profile by Protocol
TCP (0%)
UDP (0%)
ICMP (100%)
Portscan Traffic (0%)

Alert Group Maintenance | Cache & Status | Administration

BASE 1.2.7 (karen) (by **Kevin Johnson** and the **BASE Project Team**
Built on ACID by Roman Danyliw)

[Loaded in 0 seconds]

Figure 16. Screen shot of successful BASE setup.

The following commands were given to enable the BASE graphing capability. Prior to executing the following commands, we installed the php-peat-1.4.9-4.noach.rpm and php-gp-5.1.6-3.i380.rpm packages.

```
pear install Image_Color

pear install Log

pear install Numbers_Roman

pear install http://pear.php.net/get/Numbers_Words-0.13.1.tgz

pear install http://pear.php.net/get/Image_Graph-0.3.0dev4.tgz
```

e. Step 5

In this step, we wrote our own customized rule and tested the rule by simulating a D/DoS attack. Snort allows users to write their own rules as per organizational requirements. By default, all the Snort rules were found in /etc/snort/rules directory. The rules folder contained a file named “local.rules” through which the user could add customized rules. We added the following rule to this file.

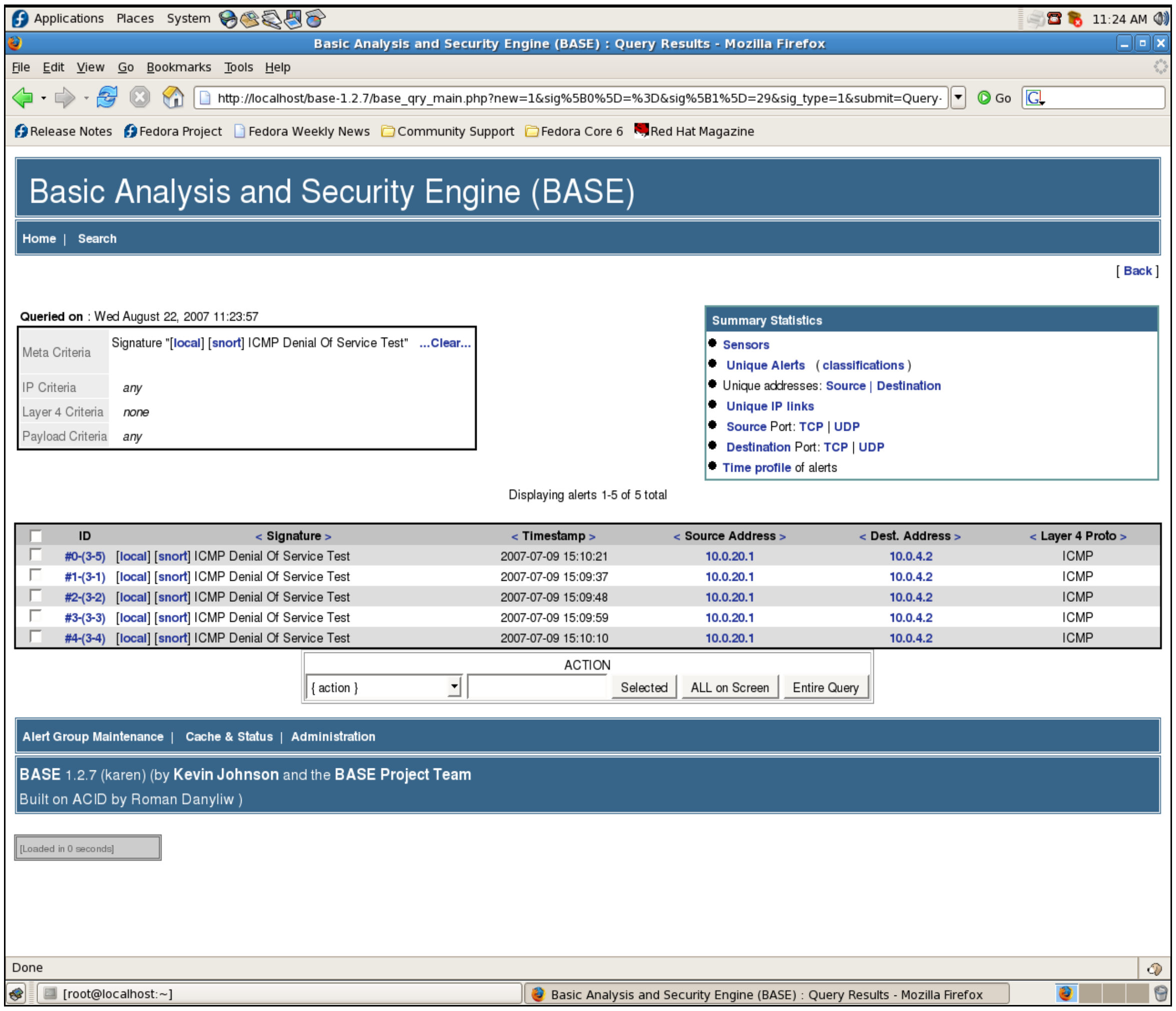
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg: “ICMP Denial of Service Test”; itype 8; classtype: misc-activity; threshold: type both, track by_dst, count 100, seconds 10; sid: 1000001; rev: 1 ;)

The above rule was written in the /etc/snort/rules/local.rules file. The detailed explanation of this rule is attached as Appendix G. After writing the rule into Snort, the following command was executed for the custom rule to take effect.

```
service snortd restart
```

f. Step 6

The next step was to test the custom rule. We simulated a D/DoS attack scenario. At this point, we will not discuss creating an attack scenario as that will be covered in the next sub-section. Rather, we would assume that the attack against the target host has commenced. The following screen shot of BASE shows that the custom rule took effect immediately.



Basic Analysis and Security Engine (BASE) : Query Results - Mozilla Firefox

http://localhost/base-1.2.7/base_qry_main.php?new=1&sig%5B0%5D=%3D&sig%5B1%5D=29&sig_type=1&submit=Query·

Basic Analysis and Security Engine (BASE)

Home | Search

[Back]

Queried on : Wed August 22, 2007 11:23:57

Meta Criteria	Signature "[local] [snort] ICMP Denial Of Service Test" ...Clear...
IP Criteria	*any*
Layer 4 Criteria	*none*
Payload Criteria	*any*

Summary Statistics
- Sensors
- Unique Alerts (classifications)
- Unique addresses: Source | Destination
- Unique IP links
- Source Port: TCP | UDP
- Destination Port: TCP | UDP
- Time profile of alerts

Displaying alerts 1-5 of 5 total

	ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
	#0-(3-5)	[local] [snort] ICMP Denial Of Service Test	2007-07-09 15:10:21	10.0.20.1	10.0.4.2	ICMP
	#1-(3-1)	[local] [snort] ICMP Denial Of Service Test	2007-07-09 15:09:37	10.0.20.1	10.0.4.2	ICMP
	#2-(3-2)	[local] [snort] ICMP Denial Of Service Test	2007-07-09 15:09:48	10.0.20.1	10.0.4.2	ICMP
	#3-(3-3)	[local] [snort] ICMP Denial Of Service Test	2007-07-09 15:09:59	10.0.20.1	10.0.4.2	ICMP
	#4-(3-4)	[local] [snort] ICMP Denial Of Service Test	2007-07-09 15:10:10	10.0.20.1	10.0.4.2	ICMP

ACTION
{ action } Selected ALL on Screen Entire Query

Alert Group Maintenance | Cache & Status | Administration

BASE 1.2.7 (karen) (by **Kevin Johnson** and the **BASE Project Team**
Built on ACID by Roman Danyliw)

[Loaded in 0 seconds]

Figure 17. Successful activation of custom rule.

The above screenshot indicates clearly that the "ICMP Denial Of Service Test" custom rule fired successfully. The destination address indicates that the victim machine was 10.0.4.2 and the source address indicates that the origin of attack was 10.0.20.1. The timestamp indicates that the threshold settings worked as desired. After recognizing more than 100 packets of type "ICMP echo request", the first rule fired at 15:09:37, then again at 15:09:48, again at 15:09:59, and so on. This proves that the threshold settings of the rule worked successfully for the D/DoS attack under consideration.

3. Instrumentation

a. Packet Capture

To monitor the packets at various locations in the network, Ethereal version 0.10.7 (C) and Wireshark version 0.99.3a Network Protocol Analyzer tools were used. Ethereal was installed on the desktop PCs running Windows XP (the attack monitoring machine and the target machine, as shown in Figure 15). Wireshark was installed on the Fedora machine.

b. System Logs/Files

Log files while setting up Snort were constantly monitored to troubleshoot the various problems faced during installation. To automate the BHR, we did not use the log files because we did not follow the search-and-destroy technique as discussed earlier.

c. Traffic Generator

To craft the ICMP packets used for the attack, CommView version 5.5 of Tamosoft and Ethereal version 0.10.7 (C) were used. Once the desired ICMP packets were crafted, the Smart Bits 6000C system, with LAN-3321A TeraMetrics XD module with two 10/100/1000 Mbps Ethernet Copper ports, was used to simulate the D/DoS attack. The in-depth explanation of this entire process is discussed in Appendix H.

d. Automation Achieved via Plug-in

An important aspect of intrusion detection is not only registering events, but also reacting to the attack attempts. Mitigating D/DoS by automating BHR is one of the reactive techniques.

As discussed previously, there are various approaches to achieve automation. Snort provides the capability to analyze data and take action based on the results. Techniques used to take action can be writing one’s own custom script, using an available plug-in, writing one’s own plug-in, etc. Snort has tools like Swatch to automate the responses of Snort alerts. Swatch is a “Simple log Watcher,” which monitors log files for specific triggers and, when any of the triggers are matched, it performs a certain action, for example, sending the route update message, sending an e-mail to system administrators about the event, etc.[37] Swatch is one approach to achieving automation. Another approach is the plug-in, which can be one of the best investments. Due to the evolution in the software field in the recent past, before writing an output plug-in, we must explore all the possibilities to find the suitable plug-in to achieve our goal.

After thorough research, it was found that a Snort has been extended with an output plug-in that notifies the SnortSam agent of blocking requests on a rule basis. SnortSam, developed by Mr. Frank Knobe (www.snortsam.net) is an intelligent agent that allows Snort to block connections by configuring firewalls or routers. SnortSam requires the Snort rule to be modified. The biggest advantage of this SnortSam agent is that it is built on the client-agent-based concept. SnortSam runs as an independent process and does not increase the workload of Snort.

To achieve the automation of BHR, the following four steps were followed:

e. Install SnortSam

Excellent help was available at www.snortsam.net to install and configure the module. The SnortSam module was installed as guided by the official site.[38] After installing the SnortSam module, the snortsam.conf file, which is located under the /usr/local/snortsam/conf directory, was configured with following options.

```
accept 10.0.4.3/24

accept localhost

logfile   /var/log/snortsam.log

daemon

cisconullroute 10.0.5.1  password  password
```

[37] Brian Caswell, Jay Beale, James C. Foster, "Snort 2.0 Intrusion Detection," Syngress, 2003, p. 304.

[38] SnortSam, http://www.snortsam.net/files/snortsam/docs/INSTALL, 5 September 2007.

The above configuration tells the SnortSam client to accept the connections from the local host as well as 10.0.4.3 (one of the interfaces of the alerter). The logfile option tells it where to log files, and, finally, “cisconullroute 10.0.5.1 password “password” tells it to use the cisconullroute plug-in. 10.0.5.1 is the address of the router where the SnortSam module will log into. The first “password” was the login password for the telnet session and the second “password” was the password to enter the configuration mode of the Cisco router.

f. Reconfigure the /etc/snort/snort.conf File

Once SnortSam was running and listening, we needed to configure Snort in two places. First, we needed to add the output plug-in so that Snort could send the block request of the destination IP address. The following command was written in the snort.conf file.

```
output alert_fwsam:  127.0.0.1
```

That command told Snort to send the blocking request to the local machine. The IP address 127.0.0.1 was configured because the SnortSam module was configured on the same machine where Snort was configured.

The second place where Snort required configuration was the rule (we wrote earlier) which was going to use the SnortSam module. This is discussed in next step.

g. Modify the Custom Rule

Now, since the output plug-in was configured, we were required to configure the Snort rules that should invoke a blocking of the destination IP address on the Cisco router. We added the “fwsam:” statement in the rule. The custom rule which we discussed in detail in Appendix G was modified as follows:

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg: “ICMP Denial of Service Test”; itype 8; classtype: misc-activity; threshold: type both, track by_dst, count 100, seconds 10; sid: 1000001; rev: 1 ; fwsam: dst, 20 minutes;)

The option “fwsam: dst, 20 minutes;” added in the rule body told Snort to invoke a block of 20 minutes on the destination address via the SnortSam module whenever the above rule fired.

After modifying the snort.conf files, we gave the following command to restart Snort.

```
service snortd restart
```

h. Modify the “ssp_cisco_nullroute.c” C File

The modified file is attached as Appendix I. This code does following three things

- It logs on the trigger router via telnet.
- It issues a command to enter the “null-route”.
- Finally, when the time interval of blocking expires, it removes the added route.

Although, the userid and password information for the telnet session is provided as discussed earlier and blocking information is provided in modified rule but the special tag value used in AS to advertise the null-route needs to be configured in this C file. Once the original code’s logic was understood, the modification was quite straightforward by adding sprintf statement. After modifying the code, the entire snortsam module needs to be rebuilt otherwise the modification will not take effect. The instructions for this process are available at the snortsam site[39]. It is worthwhile to mention, at present, this module supports only telnet sessions. In the future, this code can be rewritten in the same fashion to support the SSH protocol.

G. TESTING

In our research, network components were built gradually and tested at every stage, as shown in Figure 8. In the first stage, after configuring the routers and creating the attack scenario in the network, manual triggering was tested. After successful manual triggering, the optimum alerter was selected and fine-tuned to our requirements. A

[39] SnortSam, http://www.snortsam.net/files/snortsam/docs/INSTALL , last accessed 13 September 2007.

custom rule was made and tested on the alerter. A SnortSam module was installed on the alerter. A SnortSam plug-in was configured and its C file was modified to achieve automation of the BHR. To test the automation, a unique matrix was developed, which is shown as follows in tabular form. We will represent the results of each phase as screen shots in this section.

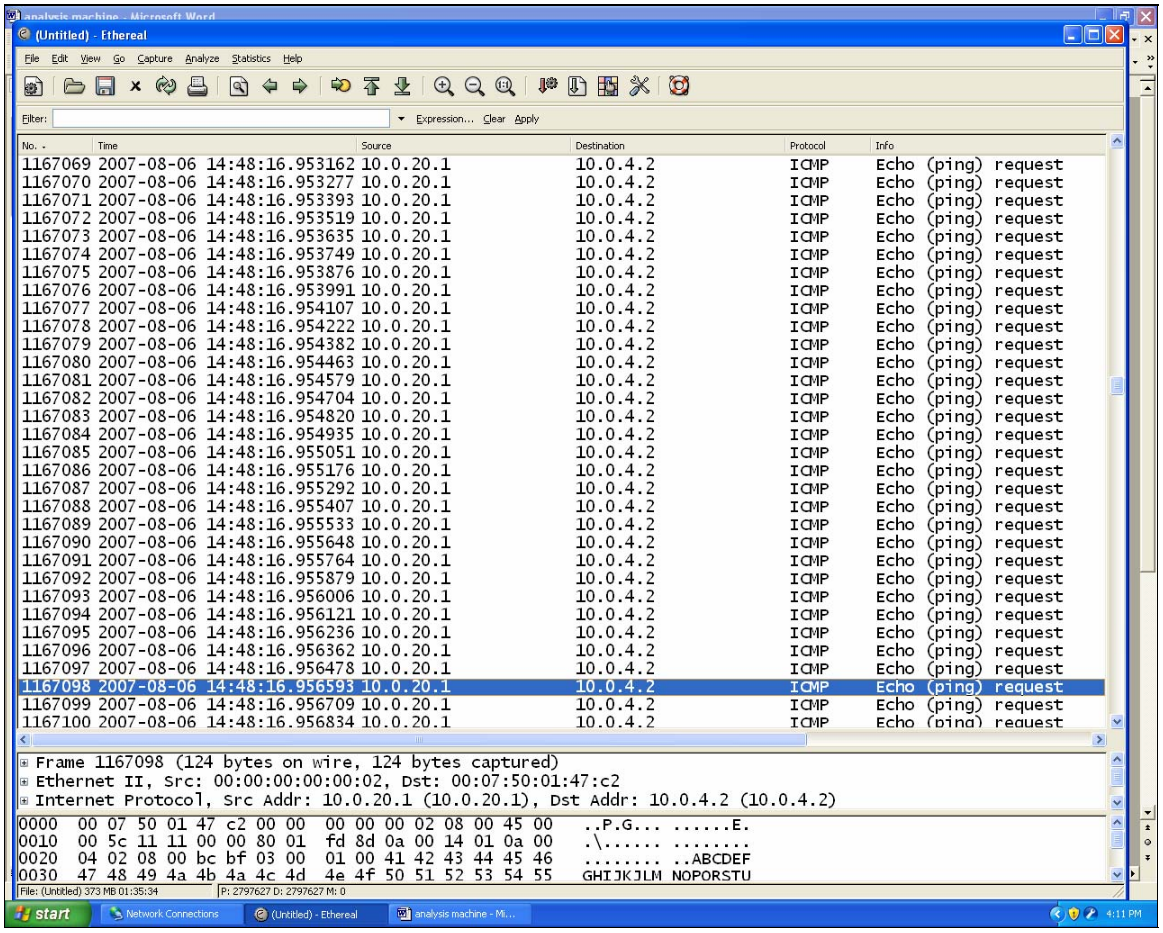
	Analysis Box (Attack Monitoring Machine)	**Victim/ Target Machine**	**IDS (Fedora Box)**	**Internal Routers**	**Trigger Router**	**Border Routers**	**Remarks**
During Attack	Packets captured (Fig 18)	Flooded with packets (Fig 19)	Packets captured (Fig 21)	Flooded	Configuration unchanged	Same route statistics	Attack started from Smart Bit.
When attack is recognized	Packets captured	Flooded with packets	Attack recognized/ Telnet session begins (Fig 22)	Flooded	Vty session (telnet starts) (Fig 22)	Same route statistics	
After recognition	Packets captured (Fig 23)	No packets (Fig 25)	Action taken (Fig 24)	No flooding	Static route added. New route advertised	New route updated	At this stage Smart Bit still generates packets

Table 4. Tabular matrix to test the automation of BHR.

Table 4 shows how the test bed network performed during the D/DoS. It has three phases: attack is underway but not yet recognized, attack is recognized by Snort, and after the recognition of attack. Throughout the testing, the network setup as delineated in Figure 15 remained the same. Explanation of these three phases is as follows:

1. During the Attack

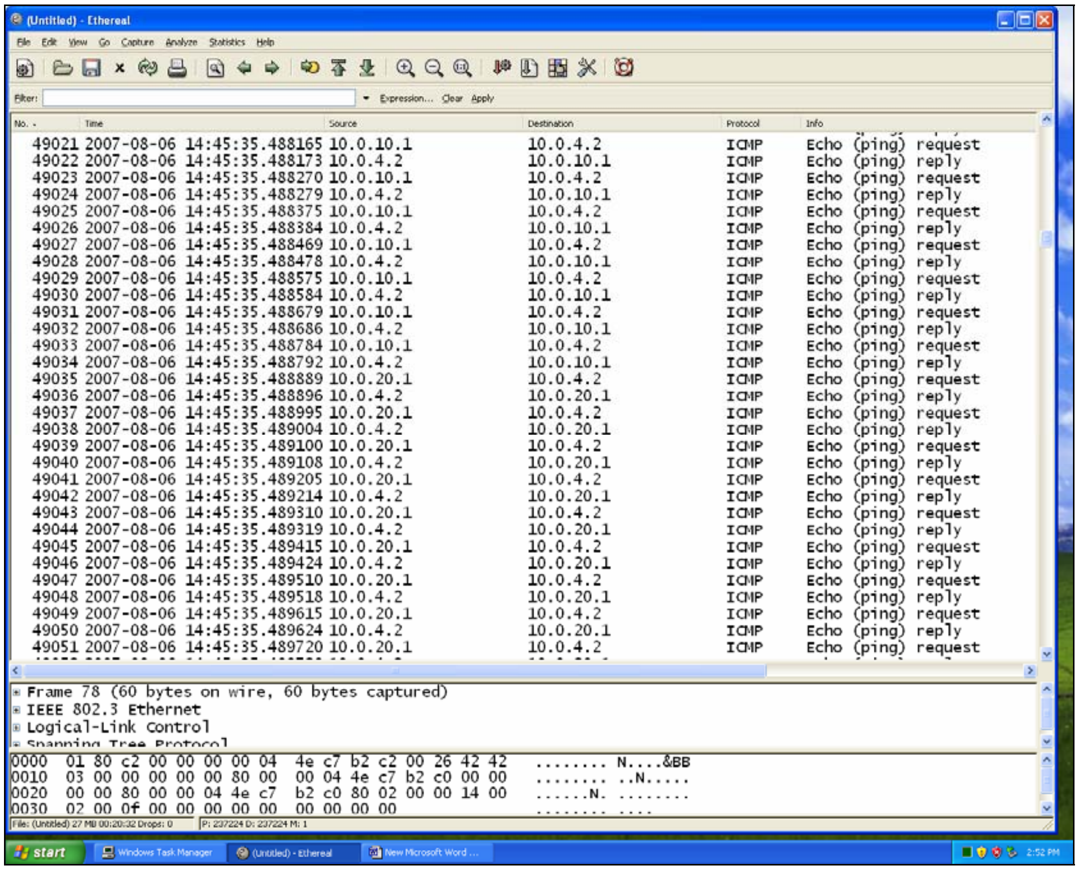
A D/DoS attack was activated from the Smart Bit 6000C system. The source of the attack was simulated from two machines with IP addresses 10.0.10.1 and 10.0.20.1. The copper ports of the LAN card of the Smart Bit 6000C systems were configured with these source addresses. When the attack started, the packets were immediately captured by Ethereal running on the attack monitoring machine. Figure 18 shows the captured packets from source IP address 10.0.20.1. As depicted in Figure 15, the attack monitoring machine was placed in network 10.0.20.0/24, therefore, packets from source 10.0.20.1 could only be captured at this machine. Hence, the screenshot displayed packets captured from IP address 10.0.20.1 only.


analysis machine - Microsoft Word
(Untitled) - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
start
Network Connections
(Untitled) - Ethereal
analysis machine - Mi...
4:11 PM

```
No. -   Time                       Source                        Destination                   Protocol  Info
1167069 2007-08-06 14:48:16.953162 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167070 2007-08-06 14:48:16.953277 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167071 2007-08-06 14:48:16.953393 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167072 2007-08-06 14:48:16.953519 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167073 2007-08-06 14:48:16.953635 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167074 2007-08-06 14:48:16.953749 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167075 2007-08-06 14:48:16.953876 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167076 2007-08-06 14:48:16.953991 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167077 2007-08-06 14:48:16.954107 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167078 2007-08-06 14:48:16.954222 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167079 2007-08-06 14:48:16.954382 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167080 2007-08-06 14:48:16.954463 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167081 2007-08-06 14:48:16.954579 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167082 2007-08-06 14:48:16.954704 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167083 2007-08-06 14:48:16.954820 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167084 2007-08-06 14:48:16.954935 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167085 2007-08-06 14:48:16.955051 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167086 2007-08-06 14:48:16.955176 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167087 2007-08-06 14:48:16.955292 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167088 2007-08-06 14:48:16.955407 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167089 2007-08-06 14:48:16.955533 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167090 2007-08-06 14:48:16.955648 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167091 2007-08-06 14:48:16.955764 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167092 2007-08-06 14:48:16.955879 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167093 2007-08-06 14:48:16.956006 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167094 2007-08-06 14:48:16.956121 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167095 2007-08-06 14:48:16.956236 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167096 2007-08-06 14:48:16.956362 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167097 2007-08-06 14:48:16.956478 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167098 2007-08-06 14:48:16.956593 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167099 2007-08-06 14:48:16.956709 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request
1167100 2007-08-06 14:48:16.956834 10.0.20.1                     10.0.4.2                      ICMP      Echo (ping) request

⊞ Frame 1167098 (124 bytes on wire, 124 bytes captured)
⊞ Ethernet II, Src: 00:00:00:00:00:02, Dst: 00:07:50:01:47:c2
⊞ Internet Protocol, Src Addr: 10.0.20.1 (10.0.20.1), Dst Addr: 10.0.4.2 (10.0.4.2)

0000   00 07 50 01 47 c2 00 00   00 00 00 02 08 00 45 00    ..P.G... ......E.
0010   00 5c 11 11 00 00 80 01   fd 8d 0a 00 14 01 0a 00    .\...... ........
0020   04 02 08 00 bc bf 03 00   01 00 41 42 43 44 45 46    ........ ..ABCDEF
0030   47 48 49 4a 4b 4a 4c 4d   4e 4f 50 51 52 53 54 55    GHIJKJLM NOPORSTU

File: (Untitled) 373 MB 01:35:34    P: 2797627 D: 2797627 M: 0
```

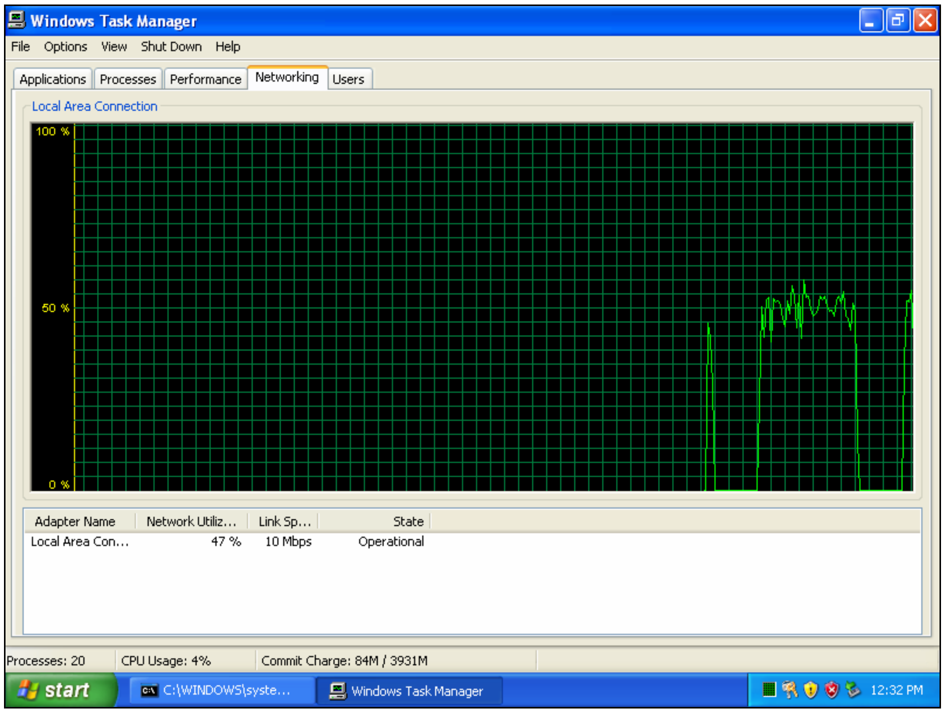
Figure 18. Packets captured at attack monitoring machine (10.0.20.3).

Packets were also captured at the target machine. Figure 19 below shows the captured packets from both sources (10.0.10.1 and 10.0.20.1). Figure 20 shows the abrupt change in the network utilization card of the target machine, which pictorially represent the effects of the D/DoS attack on the target/victim machine.



```
No. -  Time                       Source        Destination   Protocol  Info
 49021 2007-08-06 14:45:35.488165 10.0.10.1     10.0.4.2      ICMP      Echo (ping) request
 49022 2007-08-06 14:45:35.488173 10.0.4.2      10.0.10.1     ICMP      Echo (ping) reply
 49023 2007-08-06 14:45:35.488270 10.0.10.1     10.0.4.2      ICMP      Echo (ping) request
 49024 2007-08-06 14:45:35.488279 10.0.4.2      10.0.10.1     ICMP      Echo (ping) reply
 49025 2007-08-06 14:45:35.488375 10.0.10.1     10.0.4.2      ICMP      Echo (ping) request
 49026 2007-08-06 14:45:35.488384 10.0.4.2      10.0.10.1     ICMP      Echo (ping) reply
 49027 2007-08-06 14:45:35.488469 10.0.10.1     10.0.4.2      ICMP      Echo (ping) request
 49028 2007-08-06 14:45:35.488478 10.0.4.2      10.0.10.1     ICMP      Echo (ping) reply
 49029 2007-08-06 14:45:35.488575 10.0.10.1     10.0.4.2      ICMP      Echo (ping) request
 49030 2007-08-06 14:45:35.488584 10.0.4.2      10.0.10.1     ICMP      Echo (ping) reply
 49031 2007-08-06 14:45:35.488679 10.0.10.1     10.0.4.2      ICMP      Echo (ping) request
 49032 2007-08-06 14:45:35.488686 10.0.4.2      10.0.10.1     ICMP      Echo (ping) reply
 49033 2007-08-06 14:45:35.488784 10.0.10.1     10.0.4.2      ICMP      Echo (ping) request
 49034 2007-08-06 14:45:35.488792 10.0.4.2      10.0.10.1     ICMP      Echo (ping) reply
 49035 2007-08-06 14:45:35.488889 10.0.20.1     10.0.4.2      ICMP      Echo (ping) request
 49036 2007-08-06 14:45:35.488896 10.0.4.2      10.0.20.1     ICMP      Echo (ping) reply
 49037 2007-08-06 14:45:35.488995 10.0.20.1     10.0.4.2      ICMP      Echo (ping) request
 49038 2007-08-06 14:45:35.489004 10.0.4.2      10.0.20.1     ICMP      Echo (ping) reply
 49039 2007-08-06 14:45:35.489100 10.0.20.1     10.0.4.2      ICMP      Echo (ping) request
 49040 2007-08-06 14:45:35.489108 10.0.4.2      10.0.20.1     ICMP      Echo (ping) reply
 49041 2007-08-06 14:45:35.489205 10.0.20.1     10.0.4.2      ICMP      Echo (ping) request
 49042 2007-08-06 14:45:35.489214 10.0.4.2      10.0.20.1     ICMP      Echo (ping) reply
 49043 2007-08-06 14:45:35.489310 10.0.20.1     10.0.4.2      ICMP      Echo (ping) request
 49044 2007-08-06 14:45:35.489319 10.0.4.2      10.0.20.1     ICMP      Echo (ping) reply
 49045 2007-08-06 14:45:35.489415 10.0.20.1     10.0.4.2      ICMP      Echo (ping) request
 49046 2007-08-06 14:45:35.489424 10.0.4.2      10.0.20.1     ICMP      Echo (ping) reply
 49047 2007-08-06 14:45:35.489510 10.0.20.1     10.0.4.2      ICMP      Echo (ping) request
 49048 2007-08-06 14:45:35.489518 10.0.4.2      10.0.20.1     ICMP      Echo (ping) reply
 49049 2007-08-06 14:45:35.489615 10.0.20.1     10.0.4.2      ICMP      Echo (ping) request
 49050 2007-08-06 14:45:35.489624 10.0.4.2      10.0.20.1     ICMP      Echo (ping) reply
 49051 2007-08-06 14:45:35.489720 10.0.20.1     10.0.4.2      ICMP      Echo (ping) request

⊞ Frame 78 (60 bytes on wire, 60 bytes captured)
⊞ IEEE 802.3 Ethernet
⊞ Logical-Link Control
⊞ Spanning Tree Protocol

0000  01 80 c2 00 00 00 00 04  4e c7 b2 c2 00 26 42 42   ........ N....&BB
0010  03 00 00 00 00 00 80 00  00 04 4e c7 b2 c0 00 00   ........ ..N.....
0020  00 00 80 00 00 04 4e c7  b2 c0 80 02 00 00 14 00   ......N. ........
0030  02 00 0f 00 00 00 00 00  00 00 00 00               ........ ....

File: (Untitled) 27 MB 00:20:32 Drops: 0    P: 237224 D: 237224 M: 1
```

Figure 19. Packets captured at the target machine (10.0.4.2).

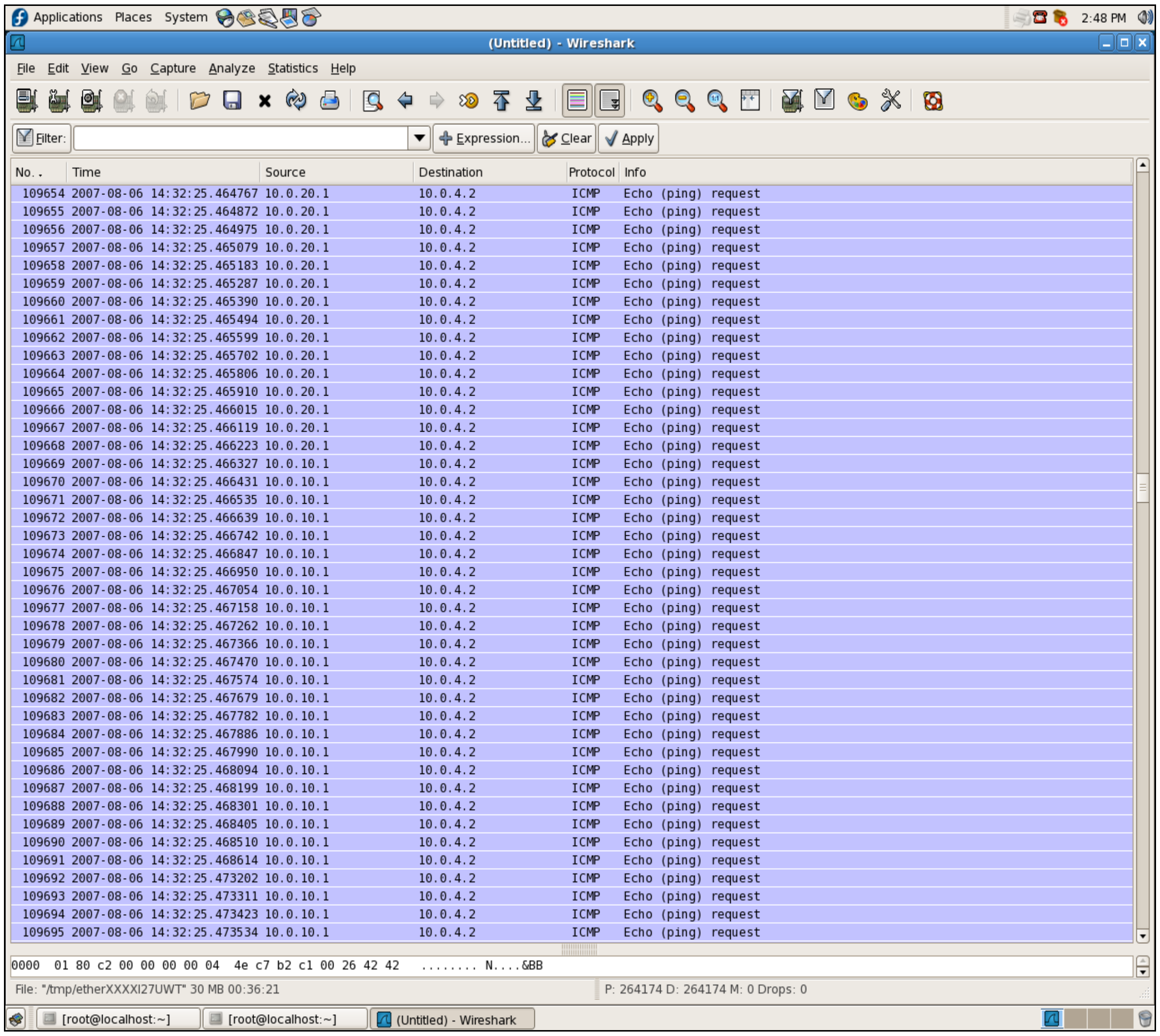


Adapter Name	Network Utiliz...	Link Sp...	State
Local Area Con...	47 %	10 Mbps	Operational

Processes: 20 CPU Usage: 4% Commit Charge: 84M / 3931M

Figure 20. Network card utilization at target machine.

Packets captured by Wireshark in the Fedora machine, where the NIDS was installed, are shown in Figure 21. The IDS detected the continuous flow of packets of type “ICMP Echo Request”. Since this sensor was placed at the entry point of the LAN (10.0.4.0/24) in promiscuous mode, both the source IP addresses were detected.



```
No. .  Time                       Source      Destination  Protocol Info
109654 2007-08-06 14:32:25.464767 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109655 2007-08-06 14:32:25.464872 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109656 2007-08-06 14:32:25.464975 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109657 2007-08-06 14:32:25.465079 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109658 2007-08-06 14:32:25.465183 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109659 2007-08-06 14:32:25.465287 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109660 2007-08-06 14:32:25.465390 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109661 2007-08-06 14:32:25.465494 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109662 2007-08-06 14:32:25.465599 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109663 2007-08-06 14:32:25.465702 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109664 2007-08-06 14:32:25.465806 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109665 2007-08-06 14:32:25.465910 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109666 2007-08-06 14:32:25.466015 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109667 2007-08-06 14:32:25.466119 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109668 2007-08-06 14:32:25.466223 10.0.20.1   10.0.4.2     ICMP     Echo (ping) request
109669 2007-08-06 14:32:25.466327 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109670 2007-08-06 14:32:25.466431 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109671 2007-08-06 14:32:25.466535 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109672 2007-08-06 14:32:25.466639 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109673 2007-08-06 14:32:25.466742 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109674 2007-08-06 14:32:25.466847 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109675 2007-08-06 14:32:25.466950 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109676 2007-08-06 14:32:25.467054 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109677 2007-08-06 14:32:25.467158 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109678 2007-08-06 14:32:25.467262 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109679 2007-08-06 14:32:25.467366 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109680 2007-08-06 14:32:25.467470 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109681 2007-08-06 14:32:25.467574 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109682 2007-08-06 14:32:25.467679 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109683 2007-08-06 14:32:25.467782 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109684 2007-08-06 14:32:25.467886 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109685 2007-08-06 14:32:25.467990 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109686 2007-08-06 14:32:25.468094 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109687 2007-08-06 14:32:25.468199 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109688 2007-08-06 14:32:25.468301 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109689 2007-08-06 14:32:25.468405 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109690 2007-08-06 14:32:25.468510 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109691 2007-08-06 14:32:25.468614 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109692 2007-08-06 14:32:25.473202 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109693 2007-08-06 14:32:25.473311 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109694 2007-08-06 14:32:25.473423 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request
109695 2007-08-06 14:32:25.473534 10.0.10.1   10.0.4.2     ICMP     Echo (ping) request

0000  01 80 c2 00 00 00 00 04  4e c7 b2 c1 00 26 42 42   ........ N....&BB
File: "/tmp/etherXXXXI27UWT" 30 MB 00:36:21          P: 264174 D: 264174 M: 0 Drops: 0
```

Figure 21. Packets captured by the Fedora box (10.0.4.3).

During the practical demonstration, it was noticed that internal routers were flooded with packets. The LED marked as "activity" on the front panel of the Cisco 2600 routers (shown as internal router 1 and internal router 2 in Figure 15) started to blink rapidly. This was a positive indication that both the routers were flooded with an enormous amount of traffic.

Route statistics were observed during the attack at the border router as well as at the trigger router. To display the current state of the routing table, Cisco routers were issued the "show ip route" command. The following is the sample output from this command, showing the current routing table in the trigger router.

Trigger-Router#show ip route

```
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
       U - per-user static route, o - ODR

Gateway of last resort is not set

S    192.168.10.0/24 is directly connected, Null0
     10.0.0.0/24 is subnetted, 3 subnets
C       10.0.8.0 is directly connected, Ethernet1/3
C       10.0.5.0 is directly connected, Ethernet1/1
B       10.0.100.0 [200/0] via 10.0.8.2, 00:00:08
```

Similarly, the following is the sample output displayed by one of the border routers when "show ip route" command was entered in the border router:

Border_Router_1#show ip route

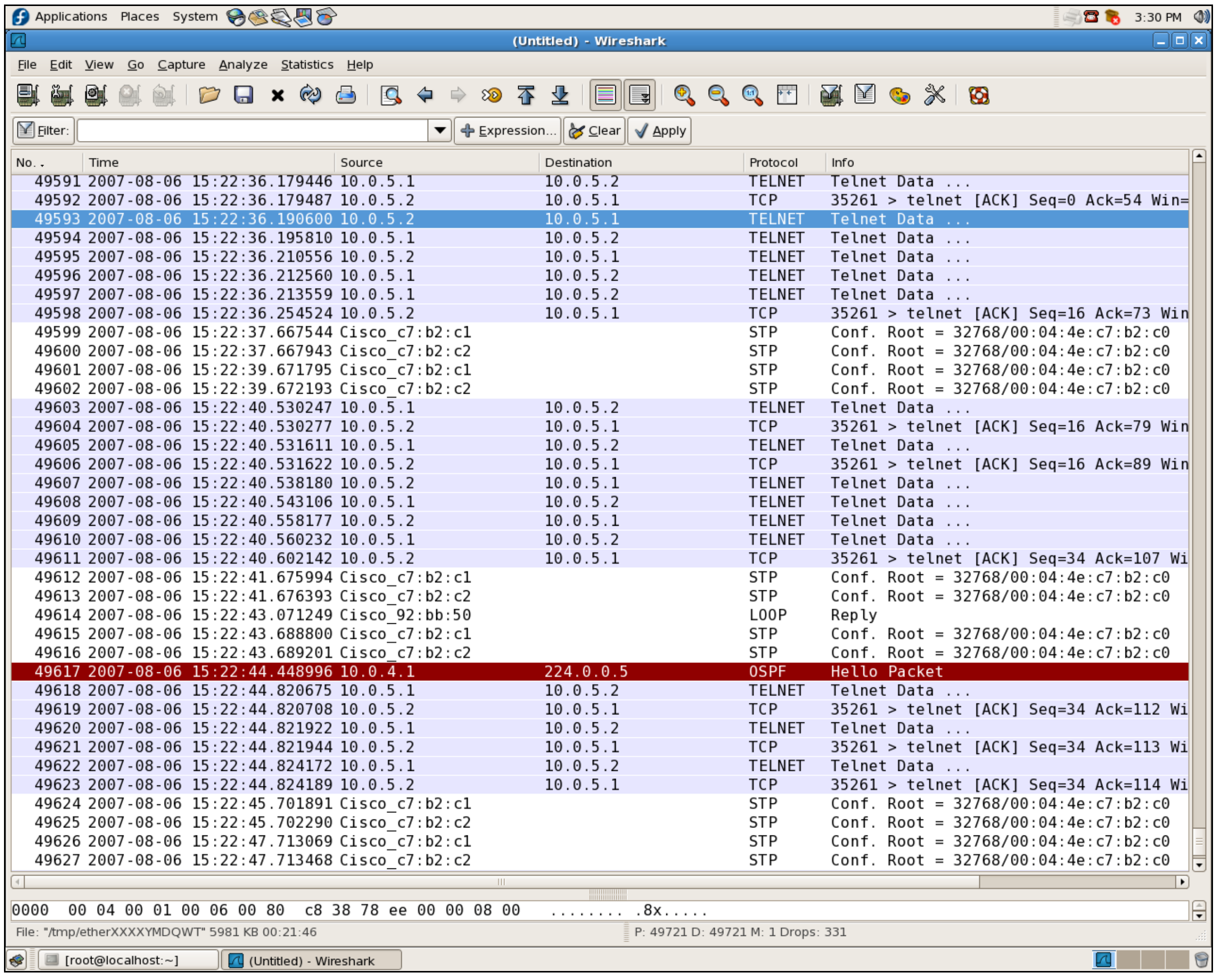
```
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

S    192.168.10.0/24 is directly connected, Null0
     10.0.0.0/24 is subnetted, 8 subnets
C       10.0.10.0 is directly connected, Ethernet0/2
O E2    10.0.8.0 [110/20] via 10.0.100.2, 00:27:27, Ethernet0/0
O       10.0.2.0 [110/20] via 10.0.100.2, 00:27:27, Ethernet0/0
                 [110/20] via 10.0.1.2, 00:27:27, Ethernet0/1
O       10.0.3.0 [110/20] via 10.0.1.2, 00:27:27, Ethernet0/1
C       10.0.1.0 is directly connected, Ethernet0/1
O       10.0.4.0 [110/30] via 10.0.1.2, 00:27:27, Ethernet0/1
O E2    10.0.20.0 [110/20] via 10.0.100.2, 00:27:28, Ethernet0/0
C       10.0.100.0 is directly connected, Ethernet0/0
Border_Router_1#
```

2. When the Attack is Recognized

During the attack recognition phase, the analysis machine and the victim machine continued to receive “ICMP echo request” packets, as shown in Figure 18 and Figure 19, respectively. Internal routers were still overwhelmed with an enormous amount of traffic. The configurations of the trigger and border routers remained unchanged.

The difference noticed in this phase was that once the attack was recognized by the alerter, the alerter fires the custom rule and starts the telnet session with the trigger router. Figure 22 pictorially represents the packets captured by Wireshark in the telnet session between the alerter (10.0.5.2) and the trigger router (10.0.5.1).

First, a three-way handshake was performed for the telnet session. Then, the alerter passes the authentication information in order to update the route information for the trigger router. The telnet session continued until the new route update message was passed.


Applications Places System
3:30 PM
(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
[root@localhost:~]
(Untitled) - Wireshark

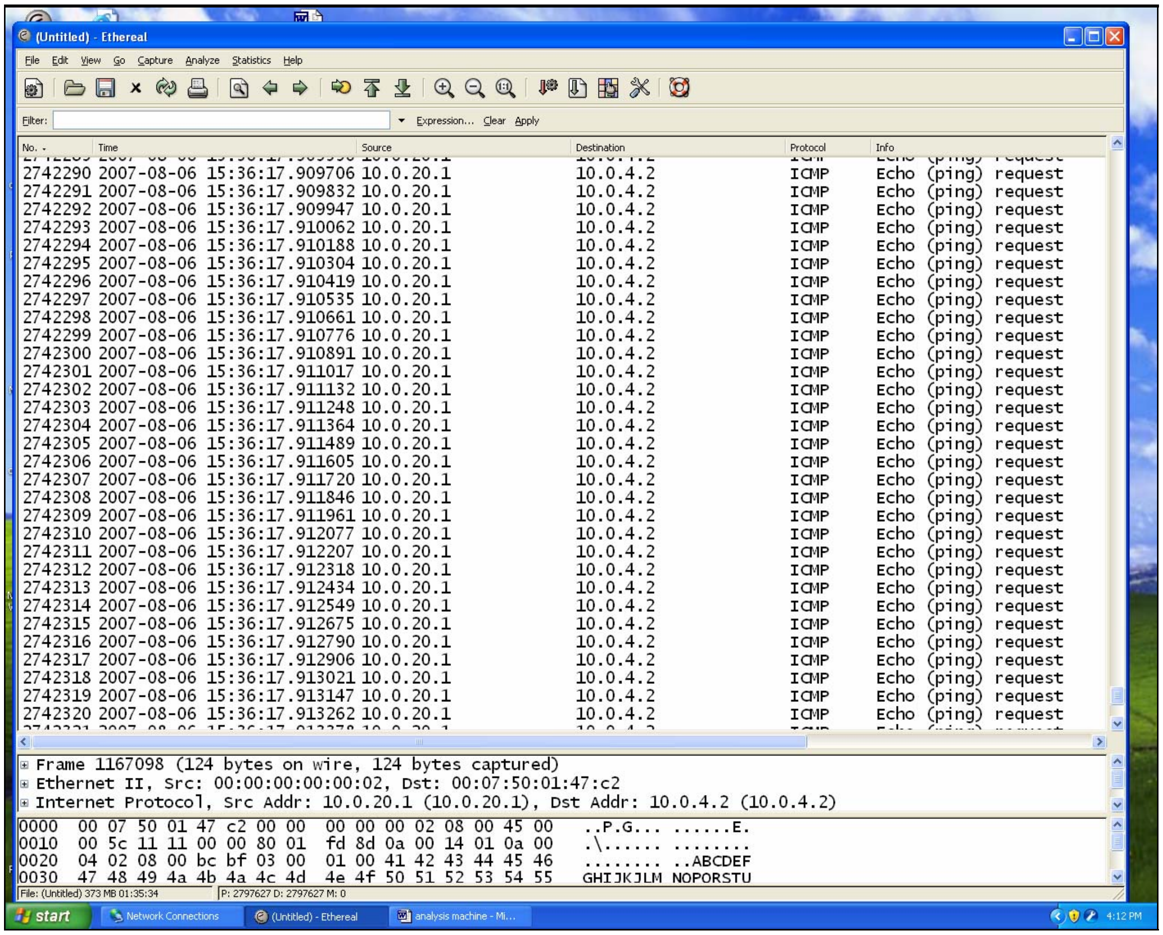
```
No. .  Time                        Source           Destination  Protocol  Info
49591 2007-08-06 15:22:36.179446 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49592 2007-08-06 15:22:36.179487 10.0.5.2         10.0.5.1     TCP       35261 > telnet [ACK] Seq=0 Ack=54 Win=
49593 2007-08-06 15:22:36.190600 10.0.5.2         10.0.5.1     TELNET    Telnet Data ...
49594 2007-08-06 15:22:36.195810 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49595 2007-08-06 15:22:36.210556 10.0.5.2         10.0.5.1     TELNET    Telnet Data ...
49596 2007-08-06 15:22:36.212560 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49597 2007-08-06 15:22:36.213559 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49598 2007-08-06 15:22:36.254524 10.0.5.2         10.0.5.1     TCP       35261 > telnet [ACK] Seq=16 Ack=73 Win
49599 2007-08-06 15:22:37.667544 Cisco_c7:b2:c1                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49600 2007-08-06 15:22:37.667943 Cisco_c7:b2:c2                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49601 2007-08-06 15:22:39.671795 Cisco_c7:b2:c1                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49602 2007-08-06 15:22:39.672193 Cisco_c7:b2:c2                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49603 2007-08-06 15:22:40.530247 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49604 2007-08-06 15:22:40.530277 10.0.5.2         10.0.5.1     TCP       35261 > telnet [ACK] Seq=16 Ack=79 Win
49605 2007-08-06 15:22:40.531611 10.0.5.2         10.0.5.1     TELNET    Telnet Data ...
49606 2007-08-06 15:22:40.531622 10.0.5.2         10.0.5.1     TCP       35261 > telnet [ACK] Seq=16 Ack=89 Win
49607 2007-08-06 15:22:40.538180 10.0.5.2         10.0.5.1     TELNET    Telnet Data ...
49608 2007-08-06 15:22:40.543106 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49609 2007-08-06 15:22:40.558177 10.0.5.2         10.0.5.1     TELNET    Telnet Data ...
49610 2007-08-06 15:22:40.560232 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49611 2007-08-06 15:22:40.602142 10.0.5.2         10.0.5.1     TCP       35261 > telnet [ACK] Seq=34 Ack=107 Wi
49612 2007-08-06 15:22:41.675994 Cisco_c7:b2:c1                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49613 2007-08-06 15:22:41.676393 Cisco_c7:b2:c2                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49614 2007-08-06 15:22:43.071249 Cisco_92:bb:50                LOOP      Reply
49615 2007-08-06 15:22:43.688800 Cisco_c7:b2:c1                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49616 2007-08-06 15:22:43.689201 Cisco_c7:b2:c2                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49617 2007-08-06 15:22:44.448996 10.0.4.1         224.0.0.5    OSPF      Hello Packet
49618 2007-08-06 15:22:44.820675 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49619 2007-08-06 15:22:44.820708 10.0.5.2         10.0.5.1     TCP       35261 > telnet [ACK] Seq=34 Ack=112 Wi
49620 2007-08-06 15:22:44.821922 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49621 2007-08-06 15:22:44.821944 10.0.5.2         10.0.5.1     TCP       35261 > telnet [ACK] Seq=34 Ack=113 Wi
49622 2007-08-06 15:22:44.824172 10.0.5.1         10.0.5.2     TELNET    Telnet Data ...
49623 2007-08-06 15:22:44.824189 10.0.5.2         10.0.5.1     TCP       35261 > telnet [ACK] Seq=34 Ack=114 Wi
49624 2007-08-06 15:22:45.701891 Cisco_c7:b2:c1                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49625 2007-08-06 15:22:45.702290 Cisco_c7:b2:c2                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49626 2007-08-06 15:22:47.713069 Cisco_c7:b2:c1                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0
49627 2007-08-06 15:22:47.713468 Cisco_c7:b2:c2                STP       Conf. Root = 32768/00:04:4e:c7:b2:c0

0000  00 04 00 01 00 06 00 80  c8 38 78 ee 00 00 08 00   ........ .8x.....

File: "/tmp/etherXXXXYMDQWT" 5981 KB 00:21:46                P: 49721 D: 49721 M: 1 Drops: 331
```

Figure 22. Telnet session between alerter and trigger router.

3. After the Attack is Recognized

This phase described the changes in the network environment once the telnet session between the alerter and the trigger router was complete. This phase also illustrates the fruitful results of the automation process.

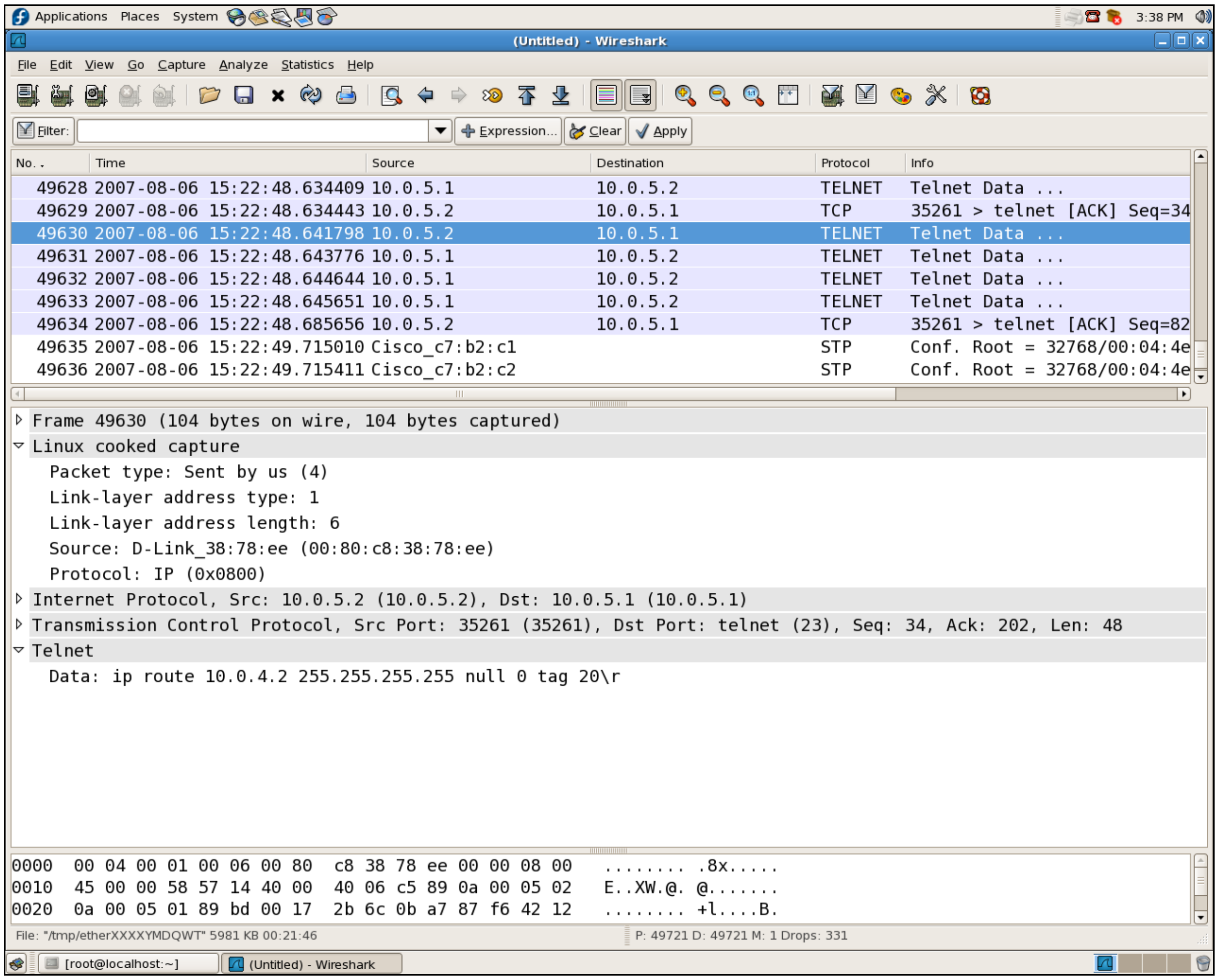
Ethereal, running on the attack monitoring box, which was connected outside the AS boundary, continued to capture packets, as shown in Figure 23. This clearly indicates that the Smart Bit System was continuously sending the “ICMP echo request” packets, which ensures that the D/DoS attack was still active.



```
No. -   Time                       Source       Destination  Protocol  Info
2742290 2007-08-06 15:36:17.909706 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742291 2007-08-06 15:36:17.909832 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742292 2007-08-06 15:36:17.909947 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742293 2007-08-06 15:36:17.910062 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742294 2007-08-06 15:36:17.910188 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742295 2007-08-06 15:36:17.910304 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742296 2007-08-06 15:36:17.910419 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742297 2007-08-06 15:36:17.910535 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742298 2007-08-06 15:36:17.910661 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742299 2007-08-06 15:36:17.910776 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742300 2007-08-06 15:36:17.910891 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742301 2007-08-06 15:36:17.911017 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742302 2007-08-06 15:36:17.911132 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742303 2007-08-06 15:36:17.911248 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742304 2007-08-06 15:36:17.911364 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742305 2007-08-06 15:36:17.911489 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742306 2007-08-06 15:36:17.911605 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742307 2007-08-06 15:36:17.911720 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742308 2007-08-06 15:36:17.911846 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742309 2007-08-06 15:36:17.911961 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742310 2007-08-06 15:36:17.912077 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742311 2007-08-06 15:36:17.912207 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742312 2007-08-06 15:36:17.912318 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742313 2007-08-06 15:36:17.912434 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742314 2007-08-06 15:36:17.912549 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742315 2007-08-06 15:36:17.912675 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742316 2007-08-06 15:36:17.912790 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742317 2007-08-06 15:36:17.912906 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742318 2007-08-06 15:36:17.913021 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742319 2007-08-06 15:36:17.913147 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
2742320 2007-08-06 15:36:17.913262 10.0.20.1    10.0.4.2     ICMP      Echo (ping) request

⊞ Frame 1167098 (124 bytes on wire, 124 bytes captured)
⊞ Ethernet II, Src: 00:00:00:00:00:02, Dst: 00:07:50:01:47:c2
⊞ Internet Protocol, Src Addr: 10.0.20.1 (10.0.20.1), Dst Addr: 10.0.4.2 (10.0.4.2)

0000  00 07 50 01 47 c2 00 00  00 00 00 02 08 00 45 00   ..P.G... ......E.
0010  00 5c 11 11 00 00 80 01  fd 8d 0a 00 14 01 0a 00   .\...... ........
0020  04 02 08 00 bc bf 03 00  01 00 41 42 43 44 45 46   ........ ..ABCDEF
0030  47 48 49 4a 4b 4a 4c 4d  4e 4f 50 51 52 53 54 55   GHIJKJLM NOPORSTU

File: (Untitled) 373 MB 01:35:34    P: 2797627 D: 2797627 M: 0
```

Figure 23. Packets captured at the attack monitoring box.

At the end of the telnet session, a new route update message, as discussed in Section C and shown in Figure 12, was passed by the alerter to the trigger router as telnet data. Figure 24 portrays the route update message, similar to Figure 12, passed during the telnet session.



```
No. .  Time                       Source        Destination  Protocol  Info
49628 2007-08-06 15:22:48.634409 10.0.5.1      10.0.5.2     TELNET    Telnet Data ...
49629 2007-08-06 15:22:48.634443 10.0.5.2      10.0.5.1     TCP       35261 > telnet [ACK] Seq=34
49630 2007-08-06 15:22:48.641798 10.0.5.2      10.0.5.1     TELNET    Telnet Data ...
49631 2007-08-06 15:22:48.643776 10.0.5.1      10.0.5.2     TELNET    Telnet Data ...
49632 2007-08-06 15:22:48.644644 10.0.5.1      10.0.5.2     TELNET    Telnet Data ...
49633 2007-08-06 15:22:48.645651 10.0.5.1      10.0.5.2     TELNET    Telnet Data ...
49634 2007-08-06 15:22:48.685656 10.0.5.2      10.0.5.1     TCP       35261 > telnet [ACK] Seq=82
49635 2007-08-06 15:22:49.715010 Cisco_c7:b2:c1              STP       Conf. Root = 32768/00:04:4e
49636 2007-08-06 15:22:49.715411 Cisco_c7:b2:c2              STP       Conf. Root = 32768/00:04:4e

Frame 49630 (104 bytes on wire, 104 bytes captured)
Linux cooked capture
  Packet type: Sent by us (4)
  Link-layer address type: 1
  Link-layer address length: 6
  Source: D-Link_38:78:ee (00:80:c8:38:78:ee)
  Protocol: IP (0x0800)
Internet Protocol, Src: 10.0.5.2 (10.0.5.2), Dst: 10.0.5.1 (10.0.5.1)
Transmission Control Protocol, Src Port: 35261 (35261), Dst Port: telnet (23), Seq: 34, Ack: 202, Len: 48
Telnet
  Data: ip route 10.0.4.2 255.255.255.255 null 0 tag 20\r

0000  00 04 00 01 00 06 00 80  c8 38 78 ee 00 00 08 00   ........ .8x.....
0010  45 00 00 58 57 14 40 00  40 06 c5 89 0a 00 05 02   E..XW.@. @.......
0020  0a 00 05 01 89 bd 00 17  2b 6c 0b a7 87 f6 42 12   ........ +l....B.

File: "/tmp/etherXXXXYMDQWT" 5981 KB 00:21:46        P: 49721 D: 49721 M: 1 Drops: 331
```

Figure 24. Route update message passed by the alerter to the trigger router.

Once the route update message is passed, the configuration of the trigger router got updated. The following was the sample output of the “show ip route” command, showing the updated routing table in the trigger router:

Trigger-Router#show ip route

```
10w3d: %SYS-5-CONFIG_I: Configured from console by vty0 (10.0.5.2)
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
U - per-user static route, o - ODR

Gateway of last resort is not set
```

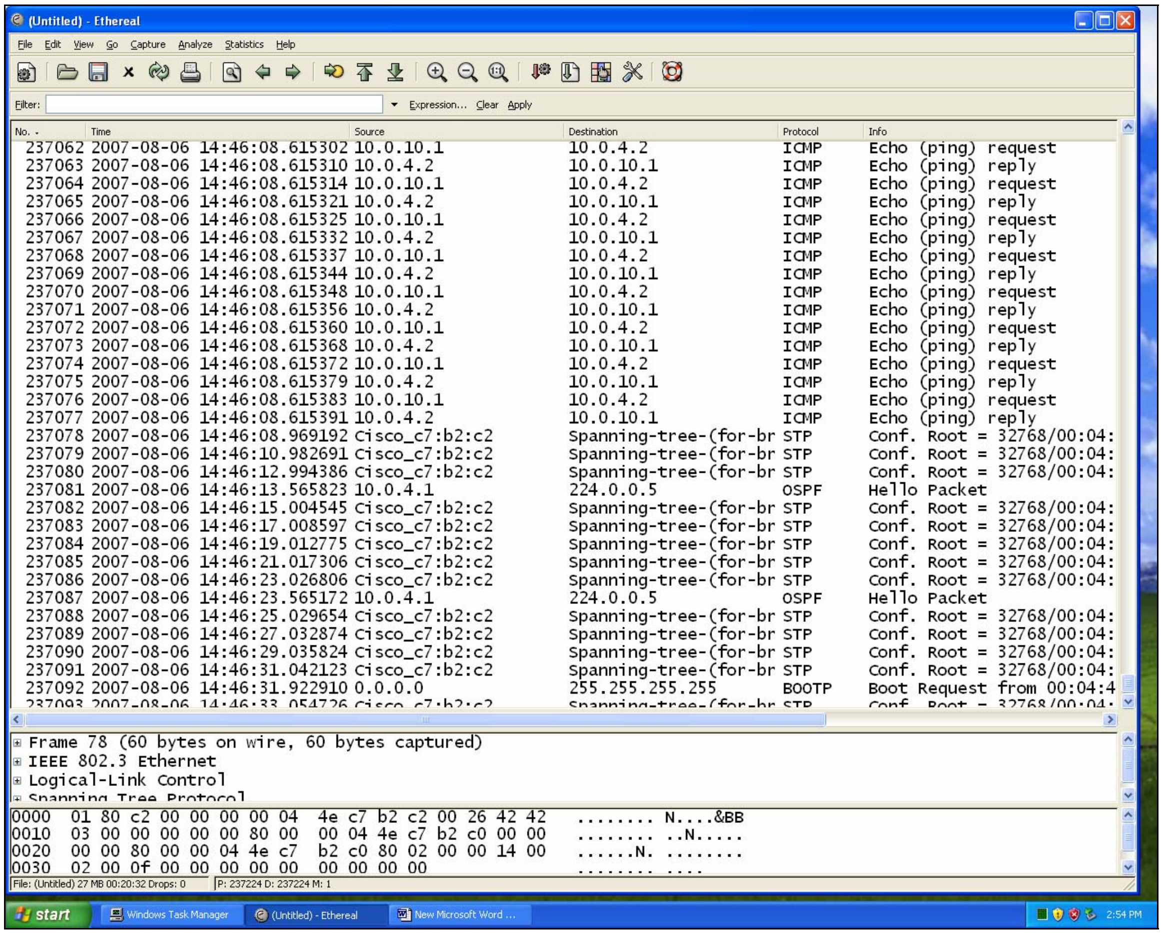
```
S    192.168.10.0/24 is directly connected, Null0
10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
C       10.0.8.0/24 is directly connected, Ethernet1/3
S       10.0.4.2/32 is directly connected, Null0
C       10.0.5.0/24 is directly connected, Ethernet1/1
B       10.0.100.0/24 [200/0] via 10.0.100.1, 00:00:55
Trigger-Router#
```

There are two noticeable things in the above sample output. First, it states that it was configured from console by vty0 (10.0.5.2), which means the configuration of the trigger router got updated by telnet session from IP address 10.0.5.2. IP address 10.0.5.2 is the address of the alerter. Second, it shows "S   10.0.4.2/32 is directly connected, Null0", which means there is a static route of IP address 10.0.4.2 to Null0 interface.

On getting this new static route information, the trigger router started the iBGP session with the border router and advertised the new route information. A sample output from the "show ip route" command, showing the current routing table in the border router is as follows:

```
Border_Router_1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

S    192.168.10.0/24 is directly connected, Null0
     10.0.0.0/8 is variably subnetted, 9 subnets, 2 masks
C       10.0.10.0/24 is directly connected, Ethernet0/2
O E2    10.0.8.0/24 [110/20] via 10.0.100.2, 00:02:57, Ethernet0/0
O       10.0.2.0/24 [110/20] via 10.0.100.2, 00:02:57, Ethernet0/0
                    [110/20] via 10.0.1.2, 00:02:57, Ethernet0/1
O       10.0.3.0/24 [110/20] via 10.0.1.2, 00:02:57, Ethernet0/1
C       10.0.1.0/24 is directly connected, Ethernet0/1
B       10.0.4.2/32 [200/0] via 192.168.10.2, 00:02:33
O       10.0.4.0/24 [110/30] via 10.0.1.2, 00:02:59, Ethernet0/1
O E2    10.0.20.0/24 [110/20] via 10.0.100.2, 00:02:59, Ethernet0/0
C       10.0.100.0/24 is directly connected, Ethernet0/0
Border_Router_1#
```

It was very evident in the sample output of the border router that a new static route to 10.0.4.2. was added via BGP protocol. Now, since the border routers were advertised with this latest route information, any D/DoS attempt against the victim machine (10.0.4.2) would be mitigated. Also, during a practical demonstration, it was noticed that activity at the internal routers returned to normal. Figure 25 indicates that Ethereal running on the victim machine stopped capturing "ICMP echo request" packets, which means that the D/DoS attack against the target machine was mitigated and the automated BHR process was successful.


(Untitled) - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply

```
No. -  Time                       Source          Destination          Protocol Info
237062 2007-08-06 14:46:08.615302 10.0.10.1       10.0.4.2             ICMP     Echo (ping) request
237063 2007-08-06 14:46:08.615310 10.0.4.2        10.0.10.1            ICMP     Echo (ping) reply
237064 2007-08-06 14:46:08.615314 10.0.10.1       10.0.4.2             ICMP     Echo (ping) request
237065 2007-08-06 14:46:08.615321 10.0.4.2        10.0.10.1            ICMP     Echo (ping) reply
237066 2007-08-06 14:46:08.615325 10.0.10.1       10.0.4.2             ICMP     Echo (ping) request
237067 2007-08-06 14:46:08.615332 10.0.4.2        10.0.10.1            ICMP     Echo (ping) reply
237068 2007-08-06 14:46:08.615337 10.0.10.1       10.0.4.2             ICMP     Echo (ping) request
237069 2007-08-06 14:46:08.615344 10.0.4.2        10.0.10.1            ICMP     Echo (ping) reply
237070 2007-08-06 14:46:08.615348 10.0.10.1       10.0.4.2             ICMP     Echo (ping) request
237071 2007-08-06 14:46:08.615356 10.0.4.2        10.0.10.1            ICMP     Echo (ping) reply
237072 2007-08-06 14:46:08.615360 10.0.10.1       10.0.4.2             ICMP     Echo (ping) request
237073 2007-08-06 14:46:08.615368 10.0.4.2        10.0.10.1            ICMP     Echo (ping) reply
237074 2007-08-06 14:46:08.615372 10.0.10.1       10.0.4.2             ICMP     Echo (ping) request
237075 2007-08-06 14:46:08.615379 10.0.4.2        10.0.10.1            ICMP     Echo (ping) reply
237076 2007-08-06 14:46:08.615383 10.0.10.1       10.0.4.2             ICMP     Echo (ping) request
237077 2007-08-06 14:46:08.615391 10.0.4.2        10.0.10.1            ICMP     Echo (ping) reply
237078 2007-08-06 14:46:08.969192 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237079 2007-08-06 14:46:10.982691 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237080 2007-08-06 14:46:12.994386 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237081 2007-08-06 14:46:13.565823 10.0.4.1        224.0.0.5            OSPF     Hello Packet
237082 2007-08-06 14:46:15.004545 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237083 2007-08-06 14:46:17.008597 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237084 2007-08-06 14:46:19.012775 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237085 2007-08-06 14:46:21.017306 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237086 2007-08-06 14:46:23.026806 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237087 2007-08-06 14:46:23.565172 10.0.4.1        224.0.0.5            OSPF     Hello Packet
237088 2007-08-06 14:46:25.029654 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237089 2007-08-06 14:46:27.032874 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237090 2007-08-06 14:46:29.035824 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237091 2007-08-06 14:46:31.042123 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:
237092 2007-08-06 14:46:31.922910 0.0.0.0         255.255.255.255      BOOTP    Boot Request from 00:04:4
237093 2007-08-06 14:46:33.054726 Cisco_c7:b2:c2  Spanning-tree-(for-br STP     Conf. Root = 32768/00:04:

⊞ Frame 78 (60 bytes on wire, 60 bytes captured)
⊞ IEEE 802.3 Ethernet
⊞ Logical-Link Control
⊞ Spanning Tree Protocol

0000  01 80 c2 00 00 00 00 04  4e c7 b2 c2 00 26 42 42   ........ N....&BB
0010  03 00 00 00 00 00 80 00  00 04 4e c7 b2 c0 00 00   ........ ..N.....
0020  00 00 80 00 00 04 4e c7  b2 c0 80 02 00 00 14 00   ......N. ........
0030  02 00 0f 00 00 00 00 00  00 00 00 00              ........ ....

File: (Untitled) 27 MB 00:20:32 Drops: 0   P: 237224 D: 237224 M: 1
```

Figure 25.	D/DoS attack against the target (10.0.4.2) was mitigated.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

Chapter IV demonstrated the real-time execution of the BHR automation process. It was noted that once an attack was detected the system close to 20 seconds to mitigate the D/DoS attack. This time includes the telnet/SSH session initiated from the IDS to the trigger router, advertisement of the null route to all the border routers, and dropping all the malicious packets at the AS boundary. In addition, the test-bed setup was monitored constantly for a period of four months and several live demonstrations were given to important visitors from the National Security Agency and other organizations. The test-bed setup was always found to be robust. No events like the IDS crashing due to application failure or excessive D/DoS traffic were noticed. The IDS never required any restarts. This leads us to the conclusion that the automation of BHR is not only an adaptable and useful technique, but it is also an efficacious and productive technique to mitigate the D/DoS attacks.

A. RECOMMENDATIONS

Though BHR cannot be the sole solution to mitigate a D/DoS attack, it is recommended that the BHR solution should be one of the mechanisms available to safeguard the target(s) and network resources from annoying D/DoS traffic within an AS. To achieve this, organizations should implement an IDS-as-BHR-alerter solution that can be customized as per local policy.

BHR proves to be one of the fastest ways to mitigate D/DoS attacks on the network. As mentioned in the conclusion, it took close to 20 seconds to mitigate a D/DoS attack. Therefore, it is recommended that if customer-triggered BHR is to be implemented as an enterprise-level security solution, then a continuous SSH session should be maintained between the customer’s IDS and the trigger router. This will significantly decrease the time needed to mitigate the D/DoS attacks (i.e., there will not be any time wasted establishing the SSH session).

It is also recommended that progressive steps should be initiated to have pre-agreements with ISPs in order to effectively defend networks via customer-triggered BHR.

B. FUTURE WORK

While this research proves that automating BHR brings substantial advantages, there are additional opportunities to produce better results. As said by Buddha in one of his inspirational quotes.[40]

All that we are is the result of what we have thought.

We also believe that the thought process should never end. Continuing on this note, we recommend that the following future studies would be very beneficial for the network security field in general and mitigation of D/DoS attacks in particular.

As discussed in Chapter IV, reserving some bandwidth in the “in-band” route path could be very productive for a customer/alerter to advertise the null route, rather than necessitating a dedicated link between customer/alerter and trigger router. Evaluating the performance of customer-triggered BHR by adopting this technique could yield some interesting results.

In this research, we used the open-source IDS Snort, which is not specially designed to mitigate D/DoS attacks. One area of future research would be develop a set of Snort rules that are specific to D/DoS attacks and sufficiently general to detect most of the known types of D/DoS attack, and, and additionally, new forms of attacks.

It may also be worthwhile to develop a smarter IDS that can be customized to trigger the source-based as well as the destination-based BHR as per policy. The following are a few examples of the characteristics which a smarter IDS might have:

[40] About, Inc., http://quotations.about.com/od/inspirationquotes/tp/10_inspiration.htm,last accessed 5 September 2007.

- If it is certain that the source IP address(es) from where the D/DoS attack originated is/are not spoofed, there should be no hesitation to block this/these source IP address(es).
- On the other hand, if it is certain that the D/DoS attack is against a host or server that is not offering any services outside the AS, then blocking the destination IP address is a better option.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. CONFIGURATION OF TRIGGER ROUTER AND DETAILED EXPLANATION

A. CONFIGURATION

```
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Trigger-Router
!
!
ip subnet-zero
!
!
!
interface Null0
 no ip unreachables
!
interface Ethernet0/0
 ip address 20.0.0.1 255.255.255.0
 no ip directed-broadcast
!
interface Ethernet1/0
 no ip address
 no ip directed-broadcast
 shutdown
!
interface Ethernet1/1
 ip address 10.0.5.1 255.255.255.0
 no ip directed-broadcast
!
interface Ethernet1/2
 no ip address
 no ip directed-broadcast
 shutdown
!
interface Ethernet1/3
 ip address 10.0.8.1 255.255.255.0
 no ip directed-broadcast
!
router bgp 209
 no synchronization
 network 10.0.8.0 mask 255.255.255.0
 redistribute static route-map StaticToBGP
 neighbor 10.0.8.2 remote-as 209
 neighbor 10.0.100.1 remote-as 209
 no auto-summary
!
ip classless
ip route 192.168.10.0 255.255.255.0 Null0
!
route-map Static permit 10
!
route-map StaticToBGP permit 10
 match tag 20
 set ip next-hop 192.168.10.2
 set local-preference 200
 set origin igp
 set community no-export
!
!
line con 0
 password password
 login
 transport input none
line aux 0
 password password
 login
line vty 0 4
 password password
 login
```

```
!
no scheduler allocate
end
```

B. DETAILED EXPLANATION

The above configuration can be explained with following figure.

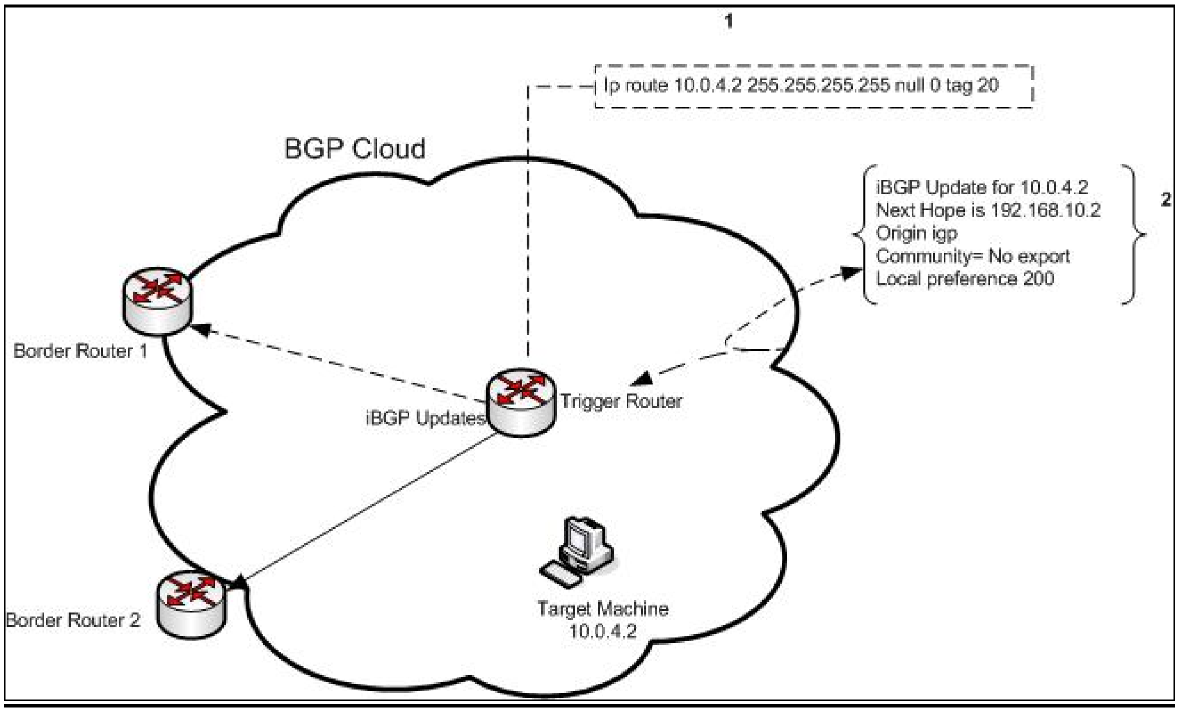


Figure 9. Target machine 10.0.4.2 under attack and trigger router sending new updates after having been informed by alerter.

In our scenario, the trigger router uses the next hop method and sets the next hop route for the destination IP address that is to be black-holed and then uses iBGP route update to propagate this route to all the edge routers (iBGP peers). The iBGP peers then set their next hop to the destination based on this update from the trigger. On every edge router, there is a static route for this next hop set to null0.[41] Upon receiving a route update for the destination IP, both the border routers in our scenario set their next hops accordingly. The static route for the next hop effectively forwards all traffic for the black-holed destination IP address to null0. The trigger router has forwarded the next hop IP

[41] Cisco, "Remotely Triggered Black Hole Filtering—Destination Based And Source Based," Cisco Press, www.cisco.com/warp/public/732/Tech/security/docs/blackhole.pdf, last accessed 9 July 2007.

address as 192.168.10.2. Now, since the border routers have already been configured at the ISP with 192.168.10.0/24 set to Null 0, therefore, all the traffic destined towards the target machine will now be dropped at the border routers. The following sequence of events will occur when the target is under attack.

1. An attack is targeted at 10.0.4.2.
2. A static route to the target IP address, 10.0.4.21, is added to the triggering router with the tag 20.
3. A route map in the trigger matches the tag 20 and sets the next hop to 192.168.10.2, origin to IGP, and community to no-export. Community no-export tells that this route should not be forwarded outside the autonomous system boundary.
4. The trigger sends the route as an iBGP route update to both the border routers.
5. The border routers that are peers receive the update and accordingly set the next hop to the target, 10.0.4.2, as 192.168.10.2.
6. Since the border routers have static routes of 192.168.10.0/24 set to null0, the final FIB entry for the target IP address (10.0.4.2) is set to null0.
7. All future traffic to 10.0.4.2 will be forwarded to null0 and dropped.

APPENDIX B. NIDS MARKET SURVEY

Characteristics	**Cisco IDS**	**NFR Sentivist**	**Intrusion Secure Net**	**Net screen-IDP**	**Snort**	**Remarks**
Network Based or Host Based	Network-Based	Network-Based	Network-Based	Network-Based	Network-Based	
Freeware	No Over $1000	No	No	No	Yes	Price of Cisco IDS.[42]
Anomaly Detection	Yes	Yes	Yes	Yes	No, but anomaly rules can be written.	Snort is signature based but simple anomaly rules can be written
Third Party Tool Integration	Yes	Yes	Yes	No	Yes	
Customizable	Yes	Yes	Yes	Yes	Yes	
ICMP Attack	Yes	Not Known	Yes	Yes	Yes	This feature is compared because in my project I have created an ICMP attack
Script Execution	Yes	No	No	No	Yes	
New Attack Patterns	Not Known	Not Known	Not Known	Not Known	No	This is not the feature which is desired by this project but this is the feature which is desired for D/DoS attack recognition.

[42] Ricky M. Magalhaes, "Host based vs. Network based IDS," http://www.windowsecurity.com/articles/Hids_vs_Nids_Part1.html,last accessed 20 July 2007.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. INTERACTION WITH ACTIVE RESEARCHER OF BRO

Question 1: Is Bro capable of detecting most of the DoS/DDoS attacks?

Reply by Bro researcher: It has a module for detecting SYN flooding, and another that performs general analysis of large bursts of traffic (but which hasn't been integrated into the rest of the system in terms of producing events upon detection).

Question 2: If not, can it be tuned to do so?

Reply: Well, its analysis is all scriptable, so you can make it do a wide range of detection. But if by “tuned” you mean there's a couple of settings that you adjust and that's it, then no, it requires scripting.

Question 3: If yes, is there any good source to learn the Bro language (e.g., a book, such as *Bro for Beginners* or *Learning Bro*).

Reply: Just the documentation that comes with it and is available from the wiki.

Question 4: Is Bro compatible with other scripts written in Python, Java, or Perl?

Reply: It can call arbitrary programs but doesn't link directly into other interpreters.

Question 5: Can Bro talk to Cisco routers? If yes, can Bro also talk to Juniper routers?

Reply: It does so, again, via calling arbitrary programs. The one we use for dynamic blocking, then ‘SSH-es’ into the LBL border router.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. INTERACTION WITH REPRESENTATIVE OF SPLUNK

Question 1: In our scenario and environment we want to integrate your product for detecting a known DDoS attack and, after detection of a particular attack, whether Splunk can handle the following: Is your product capable of talking to a Cisco router when desired to modify the running configuration of Cisco (available via plug-in or something along these lines). I can put forth further questions once I am clear that this functionality is performed by Splunk.

Reply by Splunk Representative, Mr. Michael Wilde, Sales Engineering Manager: Puri, provided you are testing Splunk 2.2 (although 3.0 is in beta now), there's a feature called “LiveSplunk” (in 3.0, it’s just called “a scheduled save search and alert”). You can run a script when results from a scheduled search have passed a threshold, for example, “number of events rises by N” triggers a script that ‘SSH-es’ into the Cisco router and makes a configuration change. While Splunk doesn't have a plug-in to modify the Cisco configuration, there's no reason why it can't trigger another mechanism.

Question 2: I think, Michael, you understood the requirement. We can run the script and it will be supported by Splunk. Now, the queries related to the earlier question are:

(A) Does Splunk have an auto feature which triggers the script (like snort) or do we need to have a separate module tied up with the alarms of Splunk?

(B) Is the daemon that listens to certain events provided by Splunk or do we need to write that daemon?

(C) What kind of scripts does Spunk support…and are Python and C part of them or not?

Reply: Puri, the answers to all of your questions are, primarily Splunk supports scripts that execute on the local operating system and are wired up for action by automated searches in the alerting (or LiveSplunk) mechanism at a minimum of one-minute intervals.

Note: The one-minute interval was the deciding factor in not using Splunk in our research.

APPENDIX E. CONFIGURATION OF BORDER ROUTER

```
!
version 12.2
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Border_Router_1
!
!
ip subnet-zero
!
!
call rsvp-sync
!
!
!
interface Null0
 no ip unreachables
!
interface Ethernet0/0
 ip address 10.0.100.1 255.255.255.0
 half-duplex
```

```
!
interface Ethernet0/1
 ip address 10.0.1.1 255.255.255.0
 half-duplex
!
interface Ethernet0/2
 ip address 10.0.10.2 255.255.255.0
 half-duplex
!
interface Ethernet0/3
 no ip address
 shutdown
 half-duplex
!
interface FastEthernet1/0
 no ip address
 shutdown
 duplex auto
 speed auto
!
router ospf 109
 log-adjacency-changes
 redistribute connected subnets
 redistribute static subnets
 redistribute bgp 209 subnets
 network 10.0.1.0 0.0.0.255 area 0
 network 10.0.100.0 0.0.0.255 area 0
!
router bgp 209
 no synchronization
 bgp log-neighbor-changes
 network 10.0.100.0 mask 255.255.255.0
 neighbor 10.0.8.1 remote-as 209
 neighbor 10.0.100.2 remote-as 209
 no auto-summary
!
ip classless
ip route 192.168.10.0 255.255.255.0 Null0
ip http server
!
!
!
dial-peer cor custom
!
!
!
!
gatekeeper
```

```
 shutdown
!
!
line con 0
 password password
 login
line aux 0
 password password
 login
line vty 0 4
 password password
 login
!
end
```

APPENDIX F. CONFIGURATION OF INTERNAL ROUTER

```
!
version 12.1

no service single-slot-reload-enable

service timestamps debug uptime

service timestamps log uptime

no service password-encryption

!

hostname Internal_Router_1

!

!

ip subnet-zero

!

!

interface Ethernet0/0

ip address 10.0.3.1 255.255.255.0

!

interface Ethernet1/0

ip address 10.0.1.2 255.255.255.0

!

interface Ethernet1/1

ip address 10.0.2.2 255.255.255.0

!
```

```
interface Ethernet1/2

no ip address

shutdown

!

interface Ethernet1/3

no ip address

shutdown

!

router ospf 109

log-adjacency-changes

network 10.0.1.0 0.0.0.255 area 0

network 10.0.2.0 0.0.0.255 area 0

network 10.0.3.0 0.0.0.255 area 0

!

ip classless

no ip http server

!

!

line con 0

password password

login

line aux 0

password password
```

```
login

line vty 0 4

password password

login

!

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. DETAILED EXPLANATION OF THE CUSTOMIZED RULE

The customized rule that was written to detect “ICMP flooding” D/DoS attacks is detailed in the following steps. This rule is written in /etc/snort/rules/local.rules.

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg: “ICMP Denial of Service Test”; itype 8; classtype: misc-activity; threshold: type both, track by_dst, count 100, seconds 10; sid: 1000001; rev: 1 ;)

The rule is a specified set of keywords and arguments used as matching criteria to identify security policy violations, known network attacks, etc.[43] Depending on the action specified in the rule, it may alert, log, ignore the packet, trigger an alert, and perform customized actions. The rule contains two logical sections: the rule header and the rule body. The rule header contains the rule’s action, protocol, source IP address, etc. The rule body contains the keywords and arguments. In our customized rule, the rule header contains following information:

alert icmp $EXTERNAL_NET any -> $HOME_NET any

(a) In our case, the rule’s action is to alert. Alert means to log the event and send an alert message to the output component.

(b) Each rule requires that we must specify the protocol. In our case, we have specified ICMP (Internet Control Message Protocol).

(c) $EXTERNAL_NET means any traffic coming from any host that is not on our internal network.

(d) “any” means any traffic coming from any port on the originating host.

(e) → is the operator.

(f) $HOME_NET means any traffic meant for a host belonging to our internal network.

(g) “any” means any traffic destined for any port to destination.

[43] Sourcefire. “Building and Operating Snort.” Security training program, 2004-2007, Sourcefire.

Snort does not actually require that there be a rule body because all the basic information is already contained in the rule header.[44] However, the rule body is the portion which has real power. The rule body allows Snort to drill into a packet. The entire rule body is enclosed in parentheses. Every rule option ends with a semicolon (even the last option). In our customized rule, the rule body contains the following information:

(msg: "ICMP Denial of Service Test"; itype 8; classtype: misc-activity; threshold: type both, track by_dst, count 100, seconds 10; sid: 1000001; rev: 1 ;)

(a) "msg" allows us to give the appropriate name to customized rule. The alert will include the message "ICMP Denial of Service Test."

(b) "itype 8' tells Snort to look for ICMP packets of the echo request type.

(c) The "classtype" is miscellaneous.

(d) We have tuned the Snort for a threshold. This option contains following information

"threshold: type both, track by dst, count 100, seconds 10"

(i) "type both" means to alert every M times we see this event during the time interval, then ignore events for the rest of the time interval.

(ii) "track by_dst" tells Snort to track the event occurring as per destination address.

(iii) "count 100" and "seconds 10" means that this rule will look for a minimum of 100 ICMP echo request packets within 10 seconds before generating an alert and then ignore the rest of the packets. For example, if there are 20,000 packets of this kind within the mentioned time duration, Snort will generate the alert on the 100th packet and will ignore the rest of the packets. Snort will start to look for this pattern again after the mentioned duration of 10 seconds expires.

[44] Sourcefire. "Building and Operating Snort." Security training program, 2004-2007, Sourcefire.

(e) The “sid” option is used to set the Snort ID to appear with the alert when the rule triggers. For custom rules, we must use a number greater than 1,000,000 to distinguish them from the rules released by Snort. In our case, since this is the only custom rule we have in Snort, a sid value of 1000001 is used.

(f) The “rev” option helps to manage rules. This option allows us to assign a revision number once a rule is edited. In our case, revision number 1 was used.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. CREATION OF ATTACK ENVIRONMENT

A ping attack occurs when an attacker attempts to overwhelm the victim's equipment through the use of ICMP echo request packets. Like most D/DoS attacks, ping attacks attempt to use CPU cycles and memory to prevent legitimate use of equipment.

The purpose of this research is to prove that the target host is partially saved from D/DoS attack once the attack pattern is recognized. The saving is "partial"; as the BHR solution takes the rather drastic action of dropping *all* traffic from sources outside of the target host's autonomous systems (AS); thereby alleviating the target host of the excessive traffic, but also rendering it incommunicado with any legitimate traffic source. Further, BHR will be ineffectual against any D/DoS traffic originating from within target host's AS. ICMP echo request packets were crafted to simulate the ping attack scenario. To craft the genuine looking ICMP echo request packet of IP ver. 4.0 we need to have a good understanding of the following:

(a) Various fields of IP ver. 4 Header.

(b) Various fields of ICMP ver. 4 Header

(c) Various fields of ICMP ver. 4 echo request Header.

The following three figures portray the information contained in the IP and ICMP headers.

1								2								3								4							
000	001	002	003	004	005	006	007	008	009	10	111	112	113	114	115	116	117	118	119	220	221	222	223	224	225	226	227	228	229	330	331
Version			**IHL**								**Type of Service**								**Total Length**												
Identification																**FFlags**			**Fragment Offset**												
Time to Live								**Protocol**								**Header Checksum**															
Source Address																															
Destination Address																															
Options																											**Padding**				

Figure 10. IP version 4 header[45]

1								2								3								4							
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3
Type								Code								Header Checksum								Header Checksum							

Figure 11. ICMPv4 Header[46]

[45] IETF Secretariat, http://www.ietf.org/rfc/rfc791.txt, last accessed 10 September 2007.

[46] Internet FAQ Archives, http://www.faqs.org/rfcs/rfc792.html,last accessed 10 September 2007.

1								2								3								4							
00 0	00 1	00 2	00 3	00 4	00 5	00 6	00 7	00 8	00 9	11 0	11 1	11 2	11 3	11 4	11 5	11 6	11 7	11 8	11 9	22 0	22 1	22 2	22 3	22 4	22 5	22 6	22 7	22 8	22 9	33 0	33 1
Identifier																**Sequence Number**															
Data																															

Figure 12. ICMP v4 Echo Request Header[47].

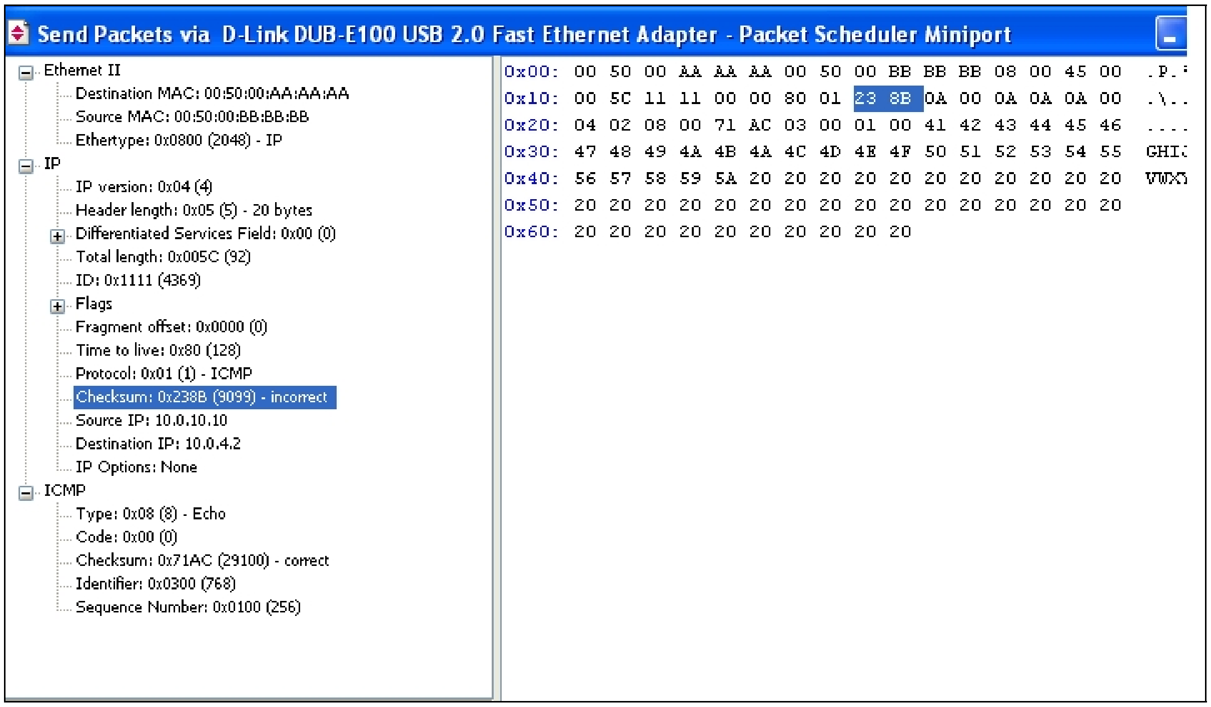
Creating the packet not only requires a network packet generator tool but also a packet capture utility, like Wireshark or Ethereal, to verify the correct checksum. Then, the hex codes of successfully crafted packets were fed into Spirent Communication’s Smart Bit Generator. The step-by-step methodology followed to craft packets is as follows:

1. Downloaded the following tools:
   (a) Ethereal (or Wireshark) to check the correct checksum.

   (b) CommView Ver 5.5 (Evaluate Version)

2. The tool CommView Version 5.5, (evaluation version) of tamosoft was used to craft ICMP packets. The combination of this tool and the packet monitoring tool, Ethereal, helped me to create genuine-looking ICMP packets in the lab. The first step was to enter the destination and source MAC address (remember, the destination MAC address is the next hop address). The following figure shows that the moment we entered the new source and destination, the IP address tool showed the checksum as incorrect.

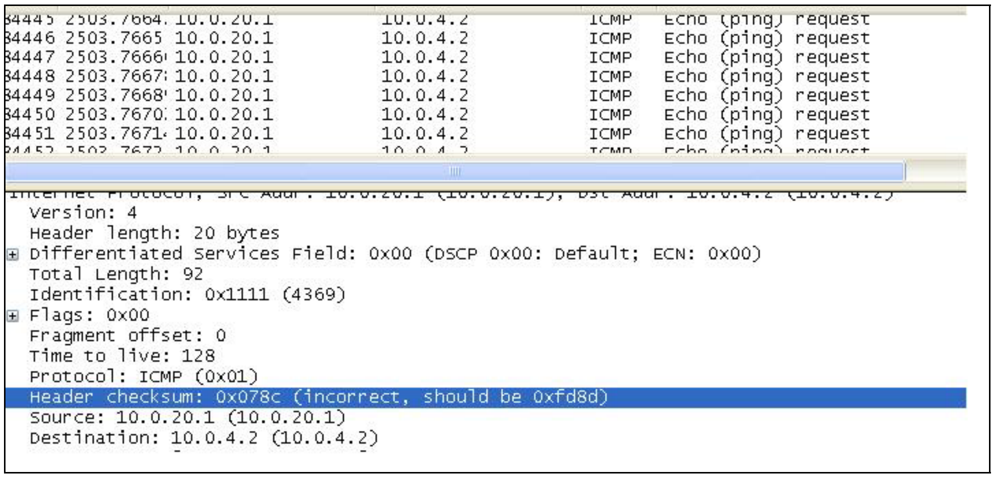
[47] Internet FAQ Archives, http://www.faqs.org/rfcs/rfc792.html,last accessed 10 September 2007.



```
Send Packets via D-Link DUB-E100 USB 2.0 Fast Ethernet Adapter - Packet Scheduler Miniport

Ethernet II
    Destination MAC: 00:50:00:AA:AA:AA
    Source MAC: 00:50:00:BB:BB:BB
    Ethertype: 0x0800 (2048) - IP
IP
    IP version: 0x04 (4)
    Header length: 0x05 (5) - 20 bytes
  + Differentiated Services Field: 0x00 (0)
    Total length: 0x005C (92)
    ID: 0x1111 (4369)
  + Flags
    Fragment offset: 0x0000 (0)
    Time to live: 0x80 (128)
    Protocol: 0x01 (1) - ICMP
    Checksum: 0x238B (9099) - incorrect
    Source IP: 10.0.10.10
    Destination IP: 10.0.4.2
    IP Options: None
ICMP
    Type: 0x08 (8) - Echo
    Code: 0x00 (0)
    Checksum: 0x71AC (29100) - correct
    Identifier: 0x0300 (768)
    Sequence Number: 0x0100 (256)

0x00:  00 50 00 AA AA AA 00 50 00 BB BB BB 08 00 45 00   .P.
0x10:  00 5C 11 11 00 00 80 01 23 8B 0A 00 0A 0A 0A 00   .\..
0x20:  04 02 08 00 71 AC 03 00 01 00 41 42 43 44 45 46   ....
0x30:  47 48 49 4A 4B 4A 4C 4D 4E 4F 50 51 52 53 54 55   GHI
0x40:  56 57 58 59 5A 20 20 20 20 20 20 20 20 20 20 20   VWX
0x50:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0x60:  20 20 20 20 20 20 20 20 20 20
```

Figure 13. Snapshot of Comm View.

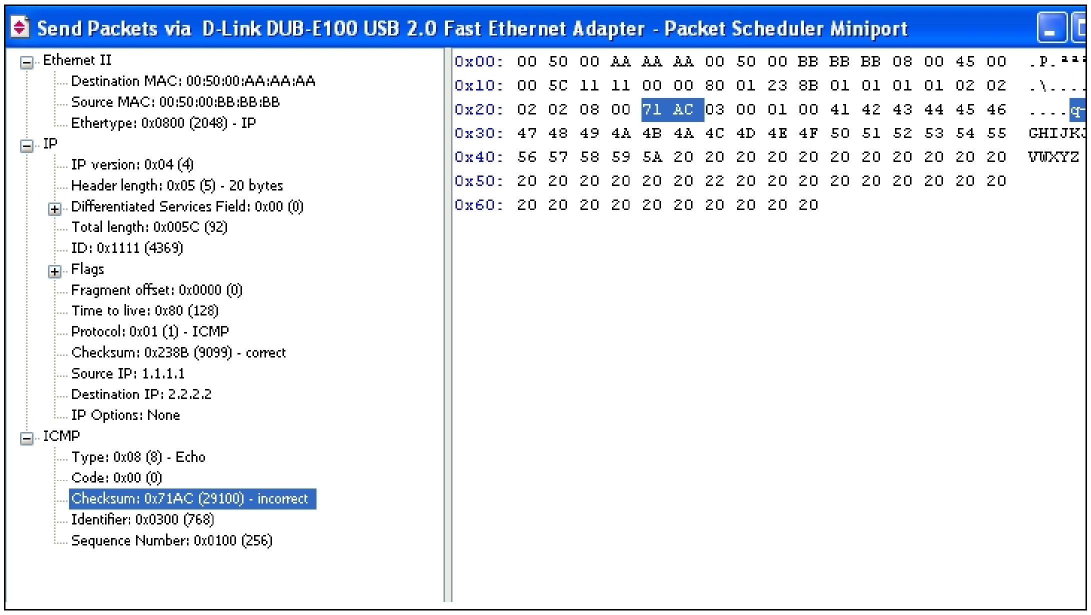
3. Still, we let the packet go with the wrong checksum and monitored the packet contents through Ethereal to check for the correct checksum. The following figure shows that Ethereal captured the packet and, not only displayed the incorrect checksum, but also suggested what the correct checksum should be. This correct checksum was then fed into NetComm to retransmit the packet.



```
34445 2503.7664 10.0.20.1        10.0.4.2        ICMP   Echo (ping) request
34446 2503.7665 10.0.20.1        10.0.4.2        ICMP   Echo (ping) request
34447 2503.7666 10.0.20.1        10.0.4.2        ICMP   Echo (ping) request
34448 2503.7667 10.0.20.1        10.0.4.2        ICMP   Echo (ping) request
34449 2503.7668 10.0.20.1        10.0.4.2        ICMP   Echo (ping) request
34450 2503.7670 10.0.20.1        10.0.4.2        ICMP   Echo (ping) request
34451 2503.7671 10.0.20.1        10.0.4.2        ICMP   Echo (ping) request
34452 2503.7672 10.0.20.1        10.0.4.2        ICMP   Echo (ping) request

Internet Protocol, Src Addr: 10.0.20.1 (10.0.20.1), Dst Addr: 10.0.4.2 (10.0.4.2)
   Version: 4
   Header length: 20 bytes
 + Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
   Total Length: 92
   Identification: 0x1111 (4369)
 + Flags: 0x00
   Fragment offset: 0
   Time to live: 128
   Protocol: ICMP (0x01)
   Header checksum: 0x078c (incorrect, should be 0xfd8d)
   Source: 10.0.20.1 (10.0.20.1)
   Destination: 10.0.4.2 (10.0.4.2)
```

Figure 14. Snapshot of Ethereal.

4. After feeding the correct header checksum, a packet was loaded with some data. This time NetComm showed the ICMP checksum to be incorrect. The same technique discussed in Steps 2 and 3 were followed to find the correct checksum. The following figures portray the detection of incorrect ICMP.



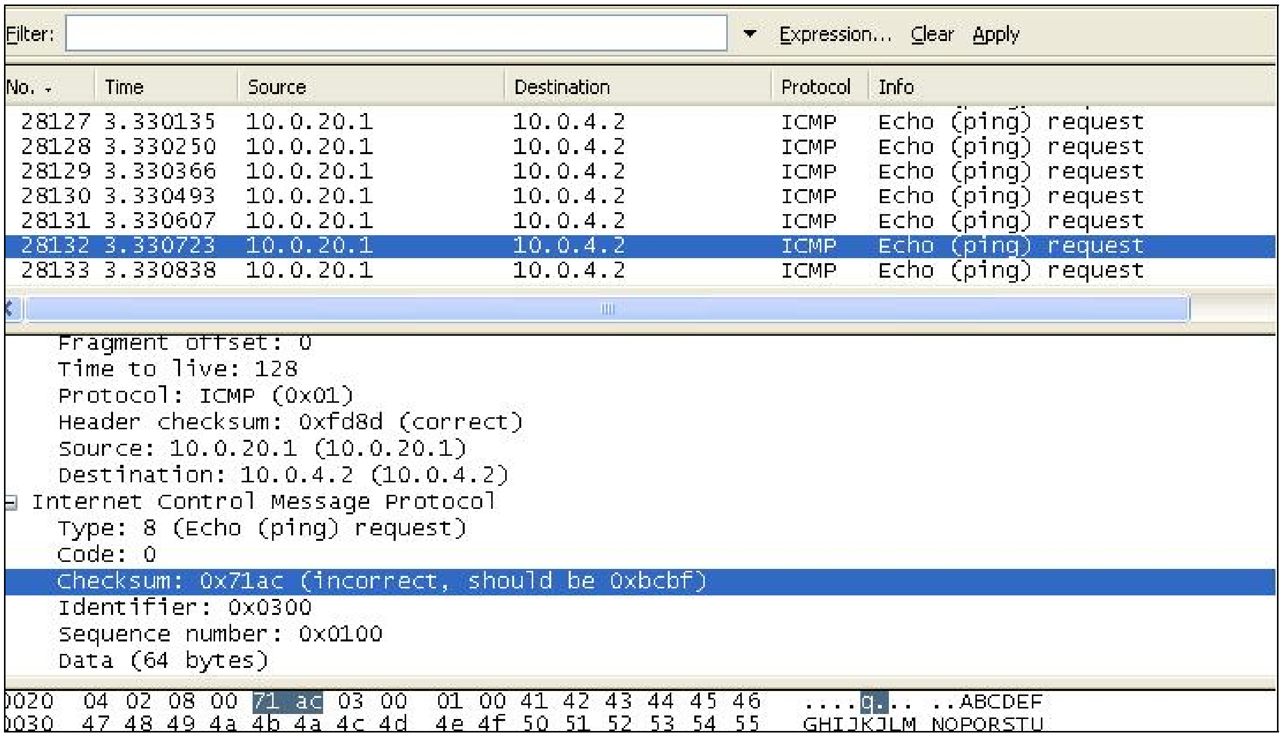
Send Packets via D-Link DUB-E100 USB 2.0 Fast Ethernet Adapter - Packet Scheduler Miniport

```
Ethernet II
    Destination MAC: 00:50:00:AA:AA:AA
    Source MAC: 00:50:00:BB:BB:BB
    Ethertype: 0x0800 (2048) - IP
IP
    IP version: 0x04 (4)
    Header length: 0x05 (5) - 20 bytes
  + Differentiated Services Field: 0x00 (0)
    Total length: 0x005C (92)
    ID: 0x1111 (4369)
  + Flags
    Fragment offset: 0x0000 (0)
    Time to live: 0x80 (128)
    Protocol: 0x01 (1) - ICMP
    Checksum: 0x238B (9099) - correct
    Source IP: 1.1.1.1
    Destination IP: 2.2.2.2
    IP Options: None
ICMP
    Type: 0x08 (8) - Echo
    Code: 0x00 (0)
    Checksum: 0x71AC (29100) - incorrect
    Identifier: 0x0300 (768)
    Sequence Number: 0x0100 (256)
```

```
0x00: 00 50 00 AA AA AA 00 50 00 BB BB BB 08 00 45 00  .P.
0x10: 00 5C 11 11 00 00 80 01 23 8B 01 01 01 01 02 02  .\....
0x20: 02 02 08 00 71 AC 03 00 01 00 41 42 43 44 45 46  ....q
0x30: 47 48 49 4A 4B 4A 4C 4D 4E 4F 50 51 52 53 54 55  GHIJK
0x40: 56 57 58 59 5A 20 20 20 20 20 20 20 20 20 20 20  VWXYZ
0x50: 20 20 20 20 20 20 22 20 20 20 20 20 20 20 20 20
0x60: 20 20 20 20 20 20 20 20 20 20
```

Figure 15. Snapshot of Comm View.

5. Ethereal detected the incorrect ICMP checksum and then suggested the correct one.



```
No.   Time      Source       Destination  Protocol  Info
28127 3.330135  10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
28128 3.330250  10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
28129 3.330366  10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
28130 3.330493  10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
28131 3.330607  10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
28132 3.330723  10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
28133 3.330838  10.0.20.1    10.0.4.2     ICMP      Echo (ping) request
```

```
    Fragment offset: 0
    Time to live: 128
    Protocol: ICMP (0x01)
    Header checksum: 0xfd8d (correct)
    Source: 10.0.20.1 (10.0.20.1)
    Destination: 10.0.4.2 (10.0.4.2)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x71ac (incorrect, should be 0xbcbf)
    Identifier: 0x0300
    Sequence number: 0x0100
    Data (64 bytes)
```

```
0020  04 02 08 00 71 ac 03 00  01 00 41 42 43 44 45 46   ....q... ..ABCDEF
0030  47 48 49 4a 4b 4a 4c 4d  4e 4f 50 51 52 53 54 55   GHIJKJLM NOPORSTU
```

Figure 16. Snapshot of Ethereal.

6. Finally, we had a genuine-looking packet. Once we sent this crafted packet, as shown as follows, we could monitor via Ethereal that there were no more errors and the destination address replied to ping requests.

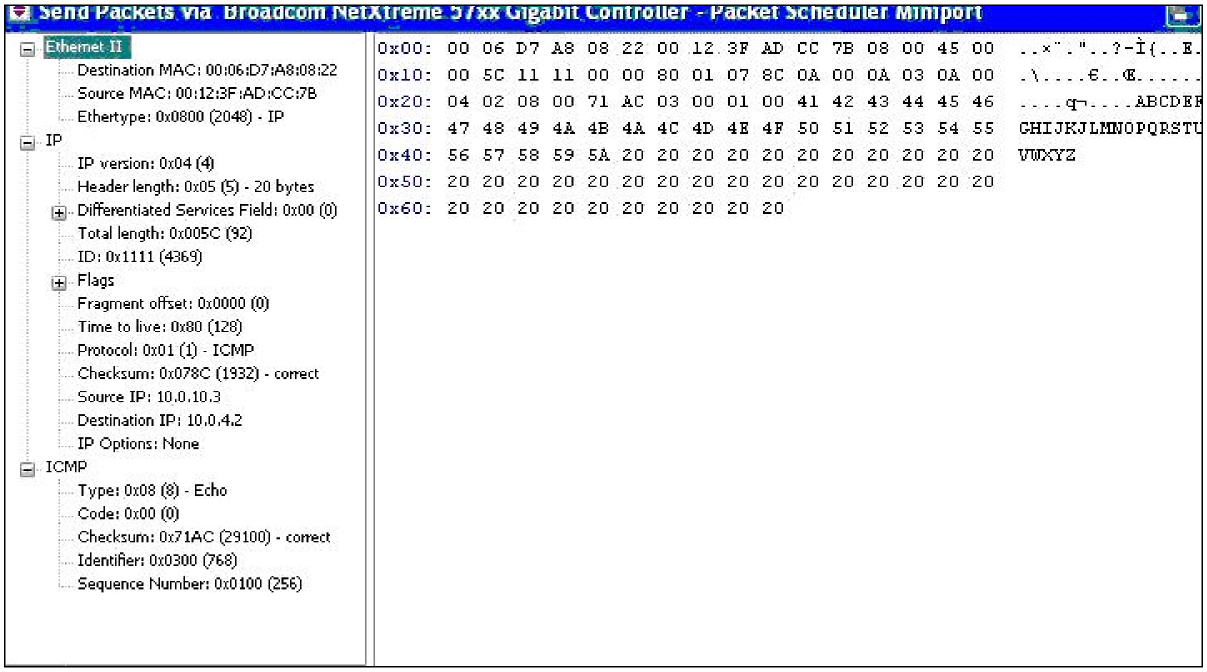

```
Send Packets via Broadcom NetXtreme 57xx Gigabit Controller - Packet Scheduler Miniport

Ethernet II
    Destination MAC: 00:06:D7:A8:08:22
    Source MAC: 00:12:3F:AD:CC:7B
    Ethertype: 0x0800 (2048) - IP
IP
    IP version: 0x04 (4)
    Header length: 0x05 (5) - 20 bytes
    Differentiated Services Field: 0x00 (0)
    Total length: 0x005C (92)
    ID: 0x1111 (4369)
    Flags
    Fragment offset: 0x0000 (0)
    Time to live: 0x80 (128)
    Protocol: 0x01 (1) - ICMP
    Checksum: 0x078C (1932) - correct
    Source IP: 10.0.10.3
    Destination IP: 10.0.4.2
    IP Options: None
ICMP
    Type: 0x08 (8) - Echo
    Code: 0x00 (0)
    Checksum: 0x71AC (29100) - correct
    Identifier: 0x0300 (768)
    Sequence Number: 0x0100 (256)

0x00:  00 06 D7 A8 08 22 00 12 3F AD CC 7B 08 00 45 00   ..×¨."..?-Ì{..E.
0x10:  00 5C 11 11 00 00 80 01 07 8C 0A 00 0A 03 0A 00   .\....€..Œ......
0x20:  04 02 08 00 71 AC 03 00 01 00 41 42 43 44 45 46   ....q¬....ABCDEF
0x30:  47 48 49 4A 4B 4A 4C 4D 4E 4F 50 51 52 53 54 55   GHIJKJLMNOPQRSTU
0x40:  56 57 58 59 5A 20 20 20 20 20 20 20 20 20 20 20   VWXYZ
0x50:  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0x60:  20 20 20 20 20 20 20 20 20 20
```

Figure 17. Snapshot of successful packet sent via Comm View.

7. CommView had another feature by which we could set the transmission rate of crafted packets. To create the D/DoS scenario, we could have three machines simply running this tool and generating the packets at a rate of 2000 packets per second.

8. Since we had the Smart Bit Chassis (manufactured by Spirent Communication) available in the lab, we utilized that equipment. The following few paragraphs describe how we configured the equipment.

> This equipment is very effective. Various network-related measurements were carried out.
>
> The hex code of the crafted packet was fed into Spirent Communication's Smart Bit Generator to activate the attack. We fed different source MAC addresses although changing the MAC address would not change the IP header checksum. Still, the ICMP checksum would vary because the Smart Bit generator had a limit on the length of contents we could feed. Applying the same technique, we sent the packet from the Smart Bit generator with a bad ICMP checksum and found the correct one through Ethereal.

9. The following few paragraphs provide step-by-step instructions on how to set a customized packet in Smart Bit generator.

(a) The Smart Bit chassis should be connected to the PC from where we are about to configure. Install the Smart Bit application software onto the PC.

(b) After the installation of Smart Bit applications, launch SmartWindow by selecting Select **Start** > **Programs** > **SmartBits SystemsApplications** > **SmartWindow**. We shall see the following window.

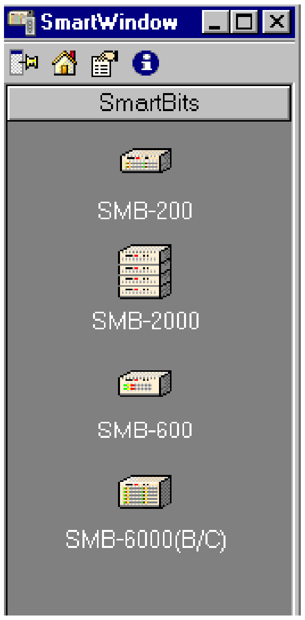


Figure 18. Snapshot of initial SmartWindow.

(c) When we select SMB-6000 followed by connect, SmartWindow will try to connect to the chassis but may not connect because we have yet to assign the IP address to our PC (ensure the physical connectivity). Also, we may not be able to view the cards connected to the Smart Bit generator. For that, we need to assign the PC an IP address belonging to the same network as the Smart Bit chassis. The default address of the Smart Bit chassis is 192.168.0.100 (/24). Therefore, we can give any address belonging to this network range. After assigning the IP address, we shall be able to ping the 192.168.0.100. Now, again click on SMB 6000, then select connect, followed by the trivial details asked by the application.

Then, you should be able to see the sample figure showing the LAN 3321A card connected to the Smart Bit generator.

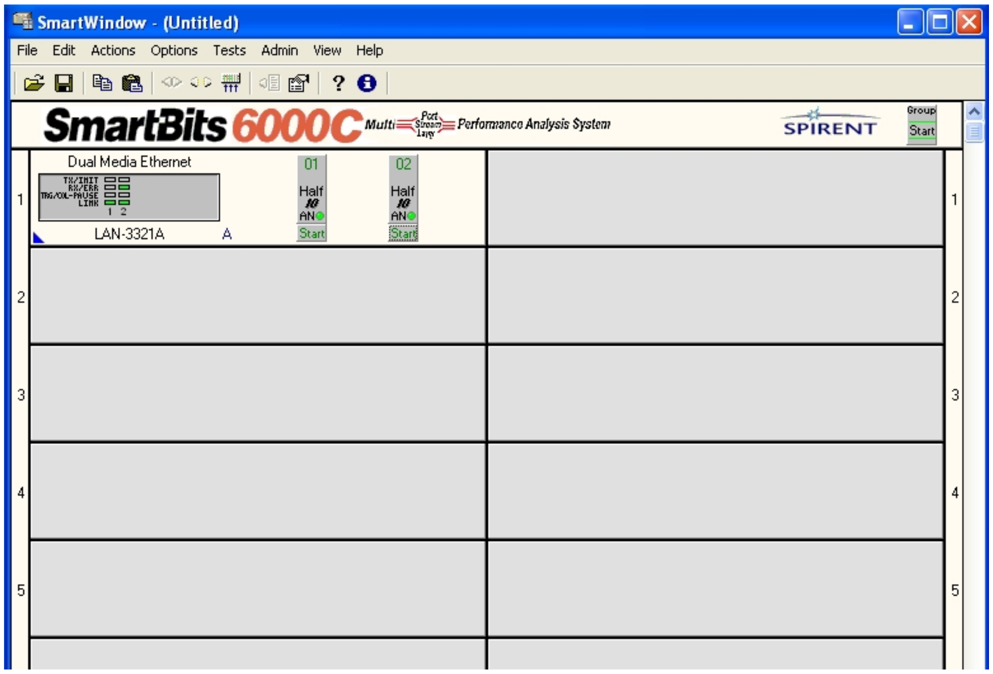
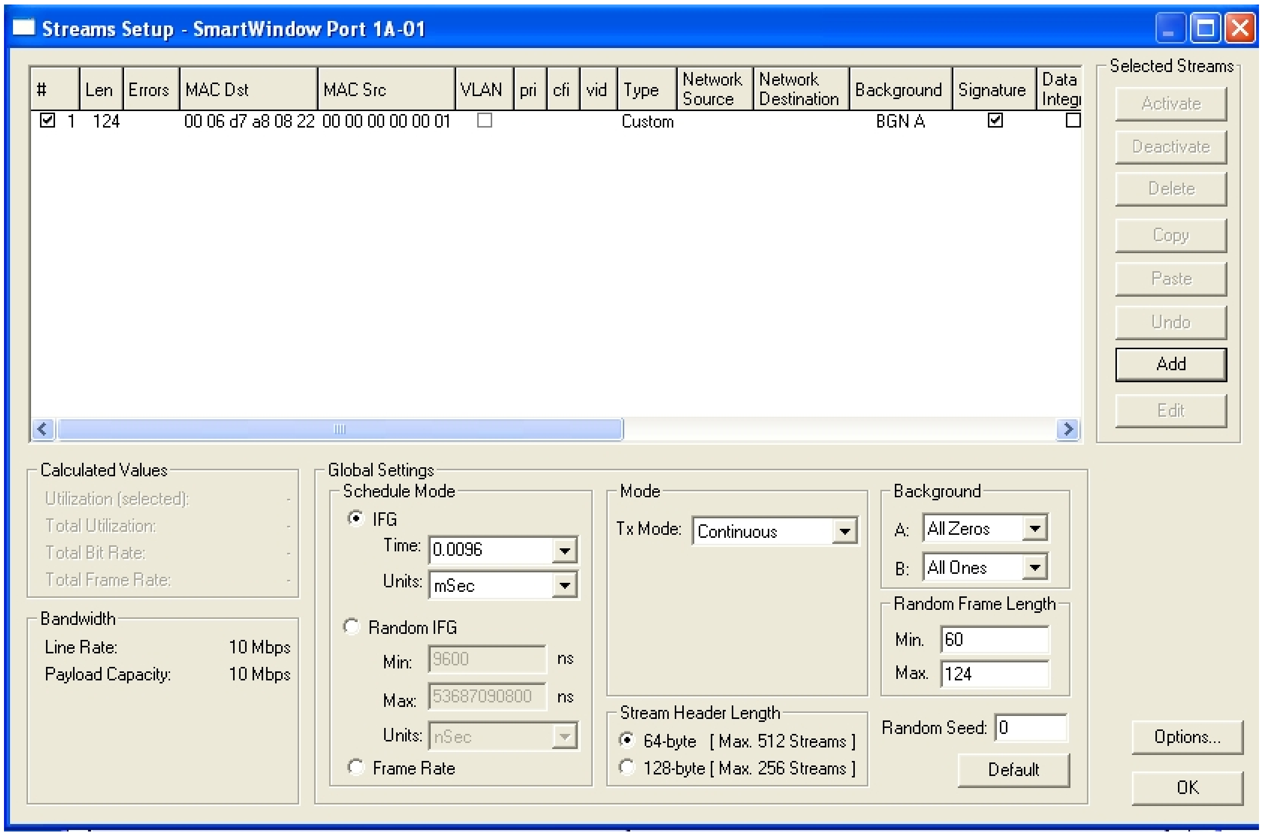

SmartWindow - (Untitled)
File Edit Actions Options Tests Admin View Help
SmartBits 6000C Multi Port Stream Layer Performance Analysis System
SPIRENT
Group Start
Dual Media Ethernet
LAN-3321A
A
01 Half 10 AN Start
02 Half 10 AN Start
1
2
3
4
5

Figure 19. Snapshot of SmartWindow after successful connection to Smart Bit chasis.

(d) Every LAN card consists of two ports. We can go to any port and right-click, then select the Transmit setup since we are going to set up the transmission of packets from these cards. Enable the smart metric mode as well as copper media because we are going to send the packets through the Ethernet ports.


Streams Setup - SmartWindow Port 1A-01

#	Len	Errors	MAC Dst	MAC Src	VLAN	pri	cfi	vid	Type	Network Source	Network Destination	Background	Signature	Data Integ
☑ 1	124		00 06 d7 a8 08 22	00 00 00 00 00 01	☐				Custom			BGN A	☑	☐

Calculated Values
Utilization (selected): -
Total Utilization: -
Total Bit Rate: -
Total Frame Rate: -

Bandwidth
Line Rate: 10 Mbps
Payload Capacity: 10 Mbps

Global Settings
Schedule Mode
● IFG
Time: 0.0096
Units: mSec
○ Random IFG
Min: 9600 ns
Max: 53687090800 ns
Units: nSec
○ Frame Rate

Mode
Tx Mode: Continuous

Stream Header Length
● 64-byte [Max. 512 Streams]
○ 128-byte [Max. 256 Streams]

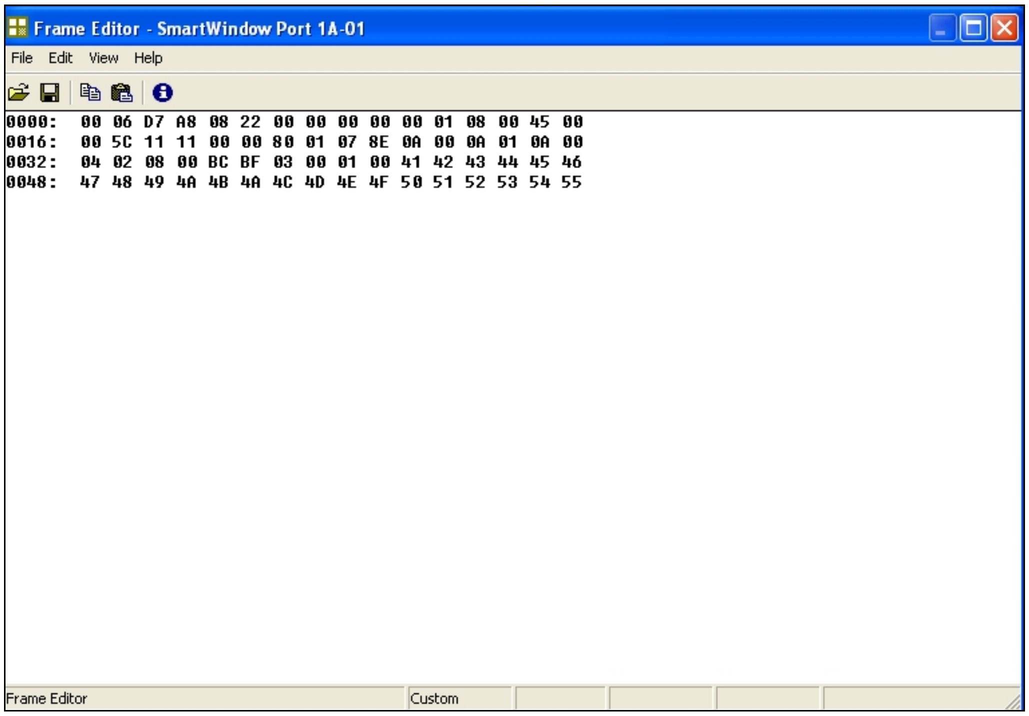
Background
A: All Zeros
B: All Ones

Random Frame Length
Min. 60
Max. 124

Random Seed: 0

Figure 20. Snapshot of transmit setup via SmartWindow.

(e) From the above figure, select the edit button and enter the custom editor, where we can feed the customized packet. A successfully delivered crafted packet through Smart Bit is shown in the next figure. Testing of this packet was already done by Comm View and Ethereal. This packet was used from port 01 of a LAN card. The other port can be used to transmit the same kind of packet by doing little bit of modification.



```
0000:  00 06 D7 A8 08 22 00 00 00 00 00 01 08 00 45 00
0016:  00 5C 11 11 00 00 80 01 07 8E 0A 00 0A 01 0A 00
0032:  04 02 08 00 BC BF 03 00 01 00 41 42 43 44 45 46
0048:  47 48 49 4A 4B 4A 4C 4D 4E 4F 50 51 52 53 54 55
```

Figure 21. Snapshot of customized packet sent through Smart Bit.

10. Since, in our scenario, the attack is from two places to simulate the D/DoS attack, we have used both the ports of the LAN 3321A card.

11. During this process, we discovered an IOS vulnerability—due to which, we had to create more genuine-looking packets. The Smart Bit generator had the built-in capability to generate the ICMP packet but, when those packets were sent as attack packets, the Cisco device receiving these specifically crafted IPv4 packets will force the inbound interface to stop processing traffic. The device may stop processing packets destined to the router, including routing protocol packets and ARP packets. No alarms will be triggered, nor will the router reload to correct itself. This issue can affect all Cisco devices running Cisco IOS software. This vulnerability was noticed when packets left the source machine successfully but did not reach the destination. While investigating, it was found via the **show interface ethernet 0/0** command in Cisco router, that the input queue size was more than the output queue. Additional investigation led us to believe that all the packets were dropped by the Cisco interface. This was verified by giving the

command **show buffers input-interface serial 0/0 packet**, which indicated millions of packets dropped. Further research showed that this vulnerability is present in all devices running Cisco IOS released before July 2003.

APPENDIX I. SSP_CISCO_NULLROUTE.C

The following is the modified source code for the Cisco null route plug-in. We must take this opportunity to thank Mr. Frank Knobbe for developing a great plug-in. His thorough research saved us precious time in not having to reinvent the wheel.

```
/* $Id: ssp_cisco_nullroute.c,v 2.2 2005/07/10 21:08:34 fknobbe Exp $
 *
 *
 * Copyright (c) 2005 Frank Knobbe <frank@knobbe.us>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
* Acknowledgements:
 *
 * Brent Erickson and Sergio Salazar for the idea and sample commands.
 *
 *
 * ssp_cisco_nullroute.c
 *
 * Purpose:
 *
 * This SnortSam plugin telnet's into one or more Cisco routers and issues
 * a route command to effectively “null-route” the intruding IP address.
 * SnortSam will remove the added routes when the blocks expire.
 *
 *
 */


#ifndef             __SSP_CISCO_NULLROUTE_C__
#define             __SSP_CISCO_NULLROUTE_C__


#include “snortsam.h”
#include “ssp_cisco_nullroute.h”


#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#ifdef WIN32
#include <winsock.h>
#else
#include <netinet/in.h>
#include <arpa/inet.h>
#endif
/* This routine parses the cisconullroute statements in the config file.
 * It builds a list of routers)
```

```
*/
void CiscoNullRouteParse(char *val,char *file,unsigned long line,DATALIST
*plugindatalist)
{       CISCONULLROUTEDATA *ciscop;
        char *p2,msg[STRBUFSIZE+2],*p3;
        struct in_addr routerip;

#ifdef FWSAMDEBUG
        printf("Debug: [cisconullroute] Plugin Parsing...\n");
#endif

        if(*val)
        {       p2=val;
                while(*p2 && !myisspace(*p2))
                        p2++;
                if(*p2)
                        *p2++ =0;
                routerip.s_addr=getip(val);
                if(routerip.s_addr)             /* If we have a valid IP address
*/
                {
        ciscop=safemalloc(sizeof(CISCONULLROUTEDATA),"ciscoparse","ciscop");
        /* create new router */
                        plugindatalist->data=ciscop;
                        ciscop->ip.s_addr=routerip.s_addr;
                        ciscop->username[0]=ciscop->enablepw[0]=ciscop-
>userlogin=0;
                        ciscop->telnetpw=ciscop->username;

                        if(*p2)
                        {       val=p2;
                                while(*val && myisspace(*val))   /* now parse the
remaining text */
                                        val++;
                                if(val)
                                {       p2=val;
                                        while(*p2 && !myisspace(*p2))
                                                p2++;
                                        if(*p2)
                                                *p2++ =0;
                                        safecopy(ciscop->username,val);   /* save
telnet password */

                                        p3=strchr(ciscop->username,'/');  /* Check if
a username is given */
                                        if(p3)
                                        {       *p3++ =0;
                                                ciscop->telnetpw=p3;
                                                ciscop->userlogin=TRUE;
                                        }

                                        if(*p2)
                        /* if we have a second password */
                                        {       while(*p2 && myisspace(*p2))
                                                        p2++;
                                                safecopy(ciscop->enablepw,p2);/* it
would be the enable password */
                                        }
                                        else
                                                safecopy(ciscop->enablepw,ciscop-
>telnetpw); /* if only one password was found, use it for both */
                                }
```

```
                    }
                    if(!ciscop->telnetpw[0])
                    {       snprintf(msg,sizeof(msg)-1,”Error: [%s: %lu] Cisco
Router defined without passwords!”,file,line);
                            logmessage(1,msg,”cisconullroute”,0);
                            free(ciscop);
                            plugindatalist->data=NULL;
                    }
#ifdef FWSAMDEBUG
                    else
                            printf(“Debug: [cisconullroute] Adding Cisco Router:
IP \”%s\”, PW \”%s\”, EN \”%s\”\n”,inettoa(ciscop->ip.s_addr),ciscop-
>telnetpw,ciscop->enablepw);
#endif
             }
             else
             {      snprintf(msg,sizeof(msg)-1,”Error: [%s: %lu] Invalid
CiscoNullRoute parameter '%s' ignored.”,file,line,val);
                    logmessage(1,msg,”cisconullroute”,0);
             }
      }
      else
      {      snprintf(msg,sizeof(msg)-1,”Error: [%s: %lu] Empty CiscoNullRoute
parameter.”,file,line);
             logmessage(1,msg,”cisconullroute”,0);
      }
}


/* This routine initiates the block. It walks the list of routers
 * telnet's in, and issues the route command.
 */
void CiscoNullRouteBlock(BLOCKINFO *bd,void *data)
{   CISCONULLROUTEDATA *ciscop;
      struct sockaddr_in thissocketaddr,routersocketaddr;
      SOCKET routersocket;
      unsigned long flag;
      char cnrmsg[STRBUFSIZE+1],cnrat[STRBUFSIZE+1];
#ifdef FWSAMDEBUG
#ifdef WIN32
      unsigned long threadid=GetCurrentThreadId();
#else
      pthread_t threadid=pthread_self();
#endif
#endif

      if(!data)
             return;
    ciscop=(CISCONULLROUTEDATA *)data;

#ifdef FWSAMDEBUG
      printf(“Debug: [cisconullroute][%lx] Plugin Blocking...\n”,(unsigned
long)threadid);
#endif

      snprintf(cnrat,sizeof(cnrat)-1,”router at %s”,inettoa(ciscop-
>ip.s_addr));

      routersocketaddr.sin_port=htons(23); /* telnet */
      routersocketaddr.sin_addr.s_addr=ciscop->ip.s_addr;
      routersocketaddr.sin_family=AF_INET;
	thissocketaddr.sin_port=htons(0); /* get a dynamic port  */
	thissocketaddr.sin_addr.s_addr=0;
	thissocketaddr.sin_family=AF_INET;

	/* create socket */
	routersocket=socket(PF_INET,SOCK_STREAM,IPPROTO_TCP);
	if(routersocket==INVALID_SOCKET)
	{	snprintf(cnrmsg,sizeof(cnrmsg)-1,”Error: [cisconullroute] Couldn't 
create socket!”);
		logmessage(1,cnrmsg,”cisconullroute”,ciscop->ip.s_addr);
		return;
	}
	/* bind it */
	if(bind(routersocket,(struct sockaddr *)&(thissocketaddr),sizeof(struct 
sockaddr)))
	{	snprintf(cnrmsg,sizeof(cnrmsg)-1,”Error: [cisconullroute] Couldn't 
bind socket!”);
		logmessage(1,cnrmsg,”ciscocnullroute”,ciscop->ip.s_addr);
		return;
	}
	/* and connect to router */
	if(connect(routersocket,(struct sockaddr 
*)&routersocketaddr,sizeof(struct sockaddr)))
	{	snprintf(cnrmsg,sizeof(cnrmsg)-1,”Error: [cisconullroute] Could 
not connect to %s! Will try later.”,cnrat);
		logmessage(1,cnrmsg,”cisconullroute”,ciscop->ip.s_addr);
	}
	else
	{	do
		{
#ifdef FWSAMDEBUG
			printf(“Debug: [cisconullroute][%lx] Connected to 
%s.\n”,(unsigned long)threadid,cnrat);
#endif
			flag=-1;
			ioctlsocket(routersocket,FIONBIO,&flag);	/* set non 
blocking  */

			if(ciscop->userlogin)
			{
	if(!sendreceive(routersocket,CNRNETWAIT,”cisconullroute”,ciscop-
>ip,””,”username”,”waiting for user logon prompt from “,cnrat))
					continue;
				snprintf(cnrmsg,sizeof(cnrmsg)-1,”%s\r”,ciscop-
>username);  /* Send username password */


	if(!sendreceive(routersocket,CNRNETWAIT,”cisconullroute”,ciscop-
>ip,cnrmsg,”pass”,”at password prompt from “,cnrat))
					continue;
				snprintf(cnrmsg,sizeof(cnrmsg)-1,”%s\r”,ciscop-
>telnetpw);  /* Send telnet password */
			}
			else
			{
	if(!sendreceive(routersocket,CNRNETWAIT,”cisconullroute”,ciscop-
>ip,””,”pass”,”waiting for logon prompt from “,cnrat))
					continue;
				snprintf(cnrmsg,sizeof(cnrmsg)-1,”%s\r”,ciscop-
>telnetpw);  /* Send telnet password */
			}
```

```
        if(!sendreceive(routersocket,CNRNETWAIT,"cisconullroute",ciscop-
>ip,cnrmsg,">“,"at logon prompt of “,cnrat))
                            continue;

                    /* Send enable */


        if(!sendreceive(routersocket,CNRNETWAIT,"cisconullroute",ciscop-
>ip,"enable\r","pass","at enable command of “,cnrat))
                            continue;

                    /* Send enable password */
                    snprintf(cnrmsg,sizeof(cnrmsg)-1,"%s\r",ciscop->enablepw);

        if(!sendreceive(routersocket,CNRNETWAIT,"cisconullroute",ciscop-
>ip,cnrmsg,"#","at enable prompt of “,cnrat))
                            continue;

                    /* Send config */

        if(!sendreceive(routersocket,CNRNETWAIT,"cisconullroute",ciscop-
>ip,"config t\r","#","at config command of “,cnrat))
                            continue;

                    /* send route command */
                    snprintf(cnrmsg,sizeof(cnrmsg)-1,"%sip route %s
255.255.255.255 null 0 tag 20\r",bd->block?"“:"no “,inettoa(bd->blockip));

        if(!sendreceive(routersocket,CNRNETWAIT,"cisconullroute",ciscop-
>ip,cnrmsg,"#","at route command of “,cnrat))
                            continue;

                    /* End input */

        if(!sendreceive(routersocket,CNRNETWAIT,"cisconullroute",ciscop-
>ip,"\032","#","at CTRL-Z of “,cnrat))
                            continue;

                    /* Save config */

        if(!sendreceive(routersocket,CNRNETWAIT,"cisconullroute",ciscop-
>ip,"write mem\r","#","at write mem command of “,cnrat))
                            continue;

                    /* and we're outta here... */

        if(!sendreceive(routersocket,CNRNETWAIT,"cisconullroute",ciscop-
>ip,"quit\r","“,"at quit command of “,cnrat))
                            continue;

              }while(FALSE);
        }
        closesocket(routersocket);
}

#endif /* __SSP_CISCO_NULLROUTE_C__ */
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

About, Inc. http://quotations.about.com/od/inspirationquotes/tp/10_inspiration.htm,last accessed 5 September 2007.

Caswell, Brian, Beale, Jay, Foster, James C. “Snort 2.0 Intrusion Detection,” Syngress, 2003, p. 304.

Cisco. “Remotely Triggered Black Hole Filtering—Destination Based and Source Based,” Cisco Press, www.cisco.com/warp/public/732/Tech/security/docs/blackhole.pdf, last accessed 9 July 2007.

Gibson, Steve. “Distributed Reflection Denial of Service,” Gibson Research Corporation, www.grc.com/dos/drdos.htm, last accessed 7 July 2007.

IETF Secretariat. http://www.ietf.org/rfc/rfc791.txt, last accessed 10 September 2007.

Institution of Electrical and Electronics Engineers. “IEEE P1220, Standard for Application and Management of the System Engineering Process,” New York, NY, 1994, p.11.

Internet FAQ Archives. http://www.faqs.org/rfcs/rfc792.html,last accessed 10 September 2007.

Kleffman, Michael D. “Analysis of Effects of BGP Black Hole Routing on a Network like the NIPRNET.” Master's Thesis, AFIT, 2005, last accessed 7 August 2007.

Lawrence Berkeley National Laboratory. http://www.bro-ids.org/, last accessed 13 September 2007.

Magalhaes, Ricky M. “Host based vs. Network based IDS,” http://www.windowsecurity.com/articles/Hids_vs_Nids_Part1.html,last accessed 20 July 2007.

Manzano, Yanet. “Tracing the Development of Denial of Service Attacks: A Corporate Analogy,” ACM Student Magazine, www.acm.org/crossroads/xrds10-1/tracingDOS.html, last accessed 7 July 2007.

Nanog. The North American Network Operators' Group, www.nanog.org/mtg-0110/ppt/greene.ppt, last accessed 8 July 2007.

Raju, Peddisetty Naga. “State-of-the-Art Intrusion Detection: Technologies, Challenges and Evaluation,” Master’s Thesis, Linkoping, 2005, last accessed 20 July 2007.

RFC 1268. www.ietf.org, last accessed 1 August 2007.

Snort.org. http://www.snort.org, last accessed 30 July 2007.

Snort.org. http://www.snort.org/community, last accessed 30 July 2007.

Snort.org. http://www.snort.org/training/, last accessed 13 September 2007.

SnortSam. http://www.snortsam.net/files/snortsam/docs/INSTALL, last accessed 13 September 2007.

Sourcefire. “Building and Operating Snort,” Security Training Program, 2004-2007.

Sourcefire. “Snort Rules,” Security Training Program, 2004-2007.

SoureForge, Inc. http://sourceforge.net/, last accessed 3 September 2007.

Stalling, William. “Network Security Essential, Third Edition,” Pearson Prentice Hall, 2007, p. 340, pp. 342-343.

Stamatelatos, Nikolaos. “A Measurement Study of BGP Black Hole Routing Performance,” Master's Thesis, September 2006.

tcpdump/libpcap. http://www.tcpdump.org/release/, last accessed 3 September 2007.

The Expect Home Page. http://expect.nist.gov/, last accessed 13 September 2007.

University of Wolverhampton. School of Computing and IT, www.scit.wlv.ac.uk/~jphb/comms/sockets.html, last accessed 2 August 2007.

Wikipedia. http://en.wikipedia.org/wiki/Bottom-up, last accessed 9 July 2007.

Wikipedia. http://en.wikipedia.org/wiki/Intrusion_detection_system, last accessed 9 July 2007.

Wikipedia. http://en.wikipedia.org/wiki/Plugin, last accessed 2 August 2007.

Wikipedia. http://en.wikipedia.org/wiki/Top-down, last accessed 9 July 2007.

Wikipedia. http://en.wikipedia.org/wiki/Anomaly-based_intrusion_detection_system, last accessed 30 July 2007.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Geoffrey Xie
   Naval Postgraduate School
   Monterey, CA

4. J.D. Fulp
   Naval Postgraduate School
   Monterey, CA

5 Vinay Puri
  Naval Postgraduate School
  Monterey, CA

6. Neal Ziring
   National Security Agency
   Fort Meade, MD

7. Terry Dossey
   National Security Agency
   Fort Meade, MD

8. C. Boger
   Naval Postgraduate School
   Monterey, CA

9. Elliott Ray
   Naval Postgraduate School
   Monterey, CA